DREAMWEAVER MX
Impariamo a realizzare un sito con stile

MULTIMEDIA
Una chat innovativa con FLASH e JSP

Supplemento a ioProgrammo n.65

Programmare il WEB

PHP
Come inviare una mail via SMTP

FLASH MX
Ora i filmati "parlano" fra loro

ASP.NET
Realizzare il libro degli ospiti con XML e ADO.NET

Lato Server
Creare la sezione protetta di un sito in ASP

PHP Nuke
Il sito dei tuoi sogni in pochi passi

Internet: azione!
Interfacce e contenuti dinamici ridisegnano il Web

SICUREZZA: Introduzione al Microsoft Security Toolkit

Programmare il WEB

Web Design

Tutorial

Lato Server

Sicurezza

STANDING OVATION (PER VOI!)

La quantità di mail e complimenti che abbiamo ricevuto per il primo numero di Programmare il Web è stata davvero stupefacente, e l'urgenza di ringraziare tutti quelli che ci hanno scritto mi costringe a superare ogni imbarazzo: un grazie di cuore a tutti!

Spero che anche questo mese non sarete delusi dai contenuti che vi offriamo e che, nei limiti dell'attuale spazio, cercano di abbracciare tutte le principali tecnologie disponibili oggi per il Web.

Per il futuro, già sento un certo prurito alle mani: l'Umts e l'integrazione fra cellulari e palmari si fanno sempre più vicini. Già oggi si vedono telefoni con display extra-large e telecamera integrata che competono per funzionalità e prestazioni con i migliori palmari di qualche anno fa e che hanno un solo problema: il software. Non tanto per le difficoltà intrinseche, quanto per la difficile interazione cui l'utente è costretto dalle piccole dimensioni del dispositivo. Le prossime sfide per i programmatori si giocheranno tutte su questo campo: trovare applicazioni di reale interesse per gli utenti e riuscire a realizzare delle interfacce valide ed efficaci anche per dispositivi "microscopici".

Molte aziende hanno pensato di superare il problema puntando tutto sui giochi. In questo modo hanno saltato a piè pari il problema della difficoltà di interazione: i primi utenti dei videogame sono sempre stati i ragazzi più giovani e, per loro, la navigazione nei più contorti menu è più un diletto che un problema...

Ovviamente, ridurre le nuove opportunità dei nuovi cellulari al semplice scopo ludico sarebbe un vero peccato. La sfida è aperta e le uniche armi ammesse sono le idee. ioProgrammo e Programmare il Web saranno sicuramente in prima linea: voi non mancate!

Raffaele del Monaco

Programmare il WEB

Supplemento ad ioProgrammo
Anno VII - N.ro 1 (65) - Gennaio 2003 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 - Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo

Dir. Editoriale Massimo Sesti
Dir. Responsabile Romina Sesti
Product Manager Antonio Meduri
Editor Gianfranco Forlino
Redazione Raffaele del Monaco, Antonio Pasqua
Collaboratori M. Battista, A. Cangiano, P. De Nictolis, C. Giustozzi, A. Perri, M. Postiglione, G. Uboldi
Segreteria di Redazione Veronica Longo

REALIZZAZIONE GRAFICA CROMATIKA Srl
C.da Lecco, zona ind. - 87030 Rende (CS)
Tel. 0984 8319 - Fax 0984 8319225
Coord. grafico: Paolo Cristiano
Coord. tecnico: Giancarlo Sicilia
Impaginazione elettronica: Aurelio Monaco, Ferdinando Gatto

PUBBLICITÀ Edizioni Master S.r.l.
Responsabile Vendite Ernesto Redaelli
Agenti Vendita Elisabetta Februo, Serenella Scarpa, Cornelio Morari

Segreteria Ufficio Vendite: Daisy Zonato
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 8321612 - Fax 02 8321764
e-mail: advertising@edmaster.it

EDITORE Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 8321482 - Fax 02 8321699
Sede di Cosenza: C.da Lecco, zona ind. - 87030 Rende (CS)
Amministratore Unico: Massimo Sesti
Responsabile Amministraz. e Finanza: Benedetto Celsa
Produzione e Logistica: Michele Carere, Daniele Ziccarelli
Diffusione: Alessandra Cervello
Marketing: Giuseppina Bruno, Leonardo Petrone, Antonio Meduri

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
tel.02 8321482
@ e-mail: servizioabbonati@edmaster.it

Stampa: Elcograf Industria Grafica (LC)
Stampa CD-Rom: KDG Italia (BZ)
Distribuzione per l'Italia: Parrini & C S.p.A. - Roma

Finito di stampare nel mese di Dicembre 2002

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, Quale Computer, DVD Magazine, Office Magazine, ioProgrammo, Linux Magazine, Softline Software World, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, Il CD-Rom di Idea Web, I Corsi di Win Magazine, Le Collection.

Dreamweaver MX: fare bella figura sul web

Con l'ultima versione del celebre prodotto della Macromedia, imporre la propria presenza su internet diventa più facile e si possono ottenere risultati di alta professionalità anche con competenze di base.

Dreamweaver MX fa parte dell'ultima generazione della suite di prodotti rilasciata dalla Macromedia, una casa produttrice di software che si è imposta sul mercato per la validità dei suoi prodotti nel campo della creatività multimediale e applicata al web.
In particolare, Dreamweaver è il tool per la gestione e produzione di pagine HTML: oltre a permettervi di creare dal nulla le pagine web più complesse gestendo in maniera user-friendly qualunque aspetto del linguaggio, dal più semplice al più complesso, l'ultimo rilascio del prodotto contiene anche tutta una serie di funzionalità per lavorare agevolmente anche con il codice di scripting lato server, consentendovi così di mettere in piedi un sito dinamico dal punto di vista del contenuto offerto con agganci a database e interazioni complesse con l'attività dell'utente (con la versione precedente tale funzionalità era disponibile solo con l'edizione UltraDev del software, mentre adesso le caratteristiche di scripting lato server sono integrate nel prodotto di base). Grazie alle sue capacità, Dreamweaver mette in grado di creare pagine dinamiche anche molto sofisticate senza presupporre vaste conoscenze di programmazione, in quanto il codice viene generato automaticamente in base a scelte semplici e guidate, e sono supportati tutti i principali linguaggi di scripting attualmente più diffusi: ASP (sia con JScript che con VBScript), PHP con mySQL, JSP, ColdFusion e persino il nuovissimo .NET con C# o VB.NET.

Fig. 1- Scelta di compatibilità con il vecchio layout delle finestre.

Relativamente all'interfaccia grafica, va detto innanzitutto che i prodotti della suite MX di Macromedia presentano una interfaccia grafica coerente contribuendo così a creare un senso di familiarità e a ridurre i tempi di apprendimento. Nel caso di Dreamweaver MX, comunque, è data all'utente la scelta tra un ambiente compatibile con la versione 4 oltre al nuovo layout stile MX: la scelta viene proposta in fase di installazione (come si vede nella **Figura 1**), ma è possibile modificare l'impostazione tramite la finestra delle preferenze (**Edit / Preferences / General / Change Workspace...**) anche ad installazione completata. Potete vedere un'immagine della nuova GUI nella **Figura 2**: come potrete notare, la filosofia dell'ambiente MX è la modularità. La finestra applicativa si compone di un'area di lavoro e tutta una serie di pannelli che possono essere spostati, ridotti a barra del titolo, raggruppati, etc. L'elenco completo dei pannelli a disposizione si trova sotto il menu Window: con **<F4>** è possibile nascondere contemporaneamente tutti i pannelli e lo stesso tasto funge da toggle per ripristinare la situazione precedente. Grazie a questo sistema di pannelli configurabili, l'utente può sistemare il proprio ambiente di lavoro come più si confà alle proprie esigenze e ai propri gusti. Nella **Figura 3** vi viene mostrato un esempio di come si possano riarrangiare i pannelli: per spostarli è sufficiente trascinare il titolo del pannello dall'estremità sinistra, per contrarli/espanderli basta fare click sul titolo, infine per ulteriori opzioni (rinominare, raggruppare, etc.) si clicca con il destro sempre sul titolo.

CREARE PAGINE WEB

Come accennavo prima, il prodotto gestisce la pagina web in tutti i suoi aspetti, entrando nel dettaglio anche dell'HTML dinamico (lato client, in questo caso) venendo incontro sia al principiante con una serie di

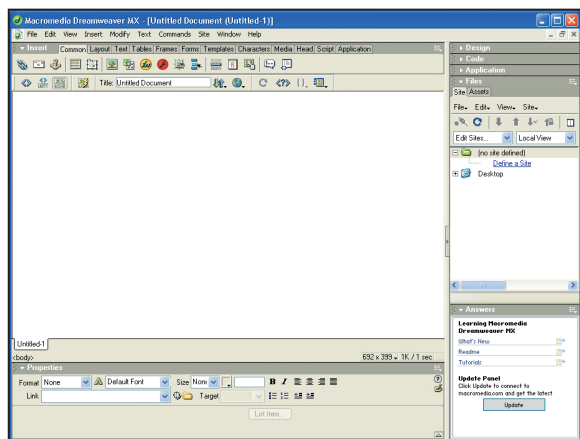



Fig. 2- la finestra di Dreamweaver con la GUI stile MX.

azioni predefinite (i behaviour) da inserire nelle pagine e da associare agli eventi scatenati dagli utenti, che ai più esperti con ausili nella stesura di codice javascript lato client.
La stesura del codice della pagina può avvenire – come nella maggior parte degli editor HTML – in maniera visuale oppure scrivendo e componendo l'HTML direttamente a mano: la scelta della modalità di lavoro avviene ad opera del menu **View** con le sottovoci **Code, Design** e **Code and Design**, oppure tramite i pulsanti corrispondenti sulla toolbar.
La **Design View** tenta di essere il più vicino possibile al

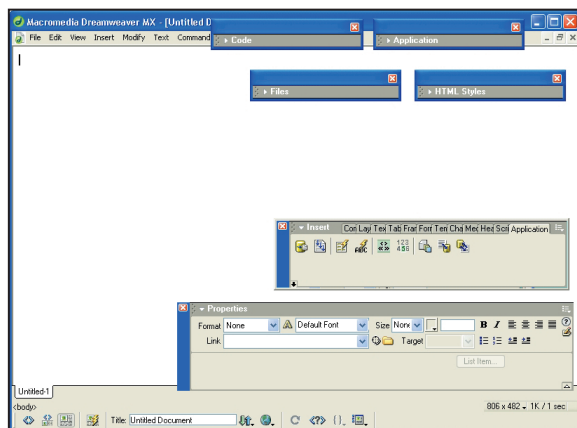



Fig. 3 - Personalizzazione dell'interfaccia grafica del prodotto.

rendering del codice HTML, ma è comunque una buona norma testare sempre il proprio lavoro su un browser vero e proprio: potete creare il vostro elenco di browser da utilizzare tramite le preferenze (**Edit / Preferences / Preview in Browser**), scegliendo ovviamente tra quelli installati sul vostro sistema.
Per creare una nuova pagina si seleziona **File / New** e si sceglie il tipo di documento da creare. Si hanno a disposizione una serie di file di base per le diverse tipologie di linguaggi e modelli che Dreamweaver gestisce, ma noi per oggi ci occupiamo di una semplice pagina HTML. È interessante notare che il prodotto è in grado di generare codice compatibile con le specifiche XHTML, cioè la versione di HTML derivata dall'XML, identica dal punto di vista sostanziale all'HTML 4.1, ma formalmente più precisa e sintatticamente più rigida: nel futuro l'HTML sarà rimpiazzato da questa nuova "veste" del linguaggio di mark-up più diffuso, quindi può essere un'idea iniziare a sviluppare i propri siti aderendo dal principio a queste specifiche (tanto, in fondo, fa tutto Dreamweaver, a patto di spuntare la check-box relativa nel dialogo di creazione del documento).
Per preparare la vostra pagina avete a disposizione vari pulsanti sulla toolbar che velocizzano il vostro lavoro e non vi costringono a ricordare lunghi elenchi di tag e loro attributi a memoria. Notate però che anche se decidete di codificare a mano la pagina, una nuova feature del prodotto viene comunque incontro alla vostra memoria: si tratta di **code hints**, simile alla capacità di molti tool di programmazione di mostrare un elenco degli attributi di oggetti e delle variabili disponibili: la caratteristica è attiva di default in maniera automatica, ma è possibile invocarla manualmente premendo contemporaneamente **<CTRL> + <SPACE>**.
La **Figura 4** mostra un esempio di aiuto per gli attributi del tag body: come vedete si tratta di una lista esaustiva che davvero velocizza e semplifica il compito di scrivere l'HTML direttamente.
Per impostare la pagina nel suo complesso potete utilizzare le proprietà di pagina (**Modify / Page Properties...** oppure **<CTRL> + J** o ancora dal menu contestuale cliccando col destro sull'area di lavoro). Si aprirà la dialog box relativa alle impostazioni di documento, dove potete dare un titolo alla pagina (tag **<TITLE>**), impostare il set di caratteri (**<META>**) ed i

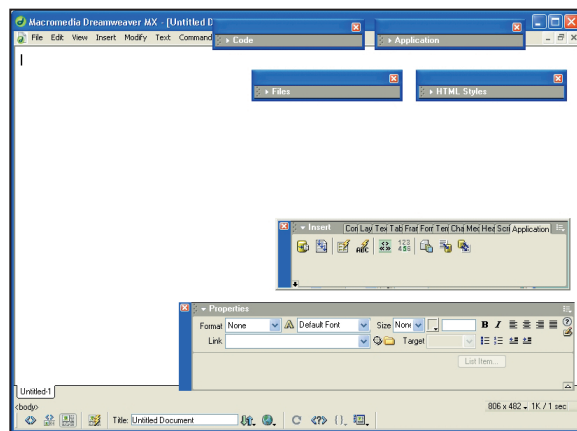



Fig. 4 - Il popup dei code hints.

DISEGNARE IL CODICE

La stesura del codice della pagina può avvenire – come nella maggior parte degli editor HTML – in maniera visuale oppure scrivendo e componendo l'HTML direttamente a mano: la scelta della modalità di lavoro avviene ad opera del menu View con le sottovoci Code, Design e Code and Design, oppure tramite i pulsanti corrispondenti sulla toolbar.

vari attributi del tag **<BODY>** (sfondo, colori dei link, etc): una possibilità interessante è la **tracing image**, un'immagine che non c'entra nulla con l'HTML e che vi viene mostrata solo in Dreamweaver per darvi la possibilità di lavorarci sopra, a mo' di ricalco. La composizione della pagina è gestita tramite la toolbar di inserimento, attraverso la quale si possono inserire i vari elementi nella **Design View**: la toolbar per lo più funziona anche in modalità **Code View**, il che può risultare comodo anche per chi preferisce scriversi la pagina in stile amanuense. Basta cliccare su uno dei pulsanti per attivare la finestra di dialogo che raccoglie tutte le informazioni necessarie per la creazione del tag.

Fig. 5 - Le proprietà della pagina web.

Si possono creare in questo modo dei link, delle immagini, delle tabelle, dei frame in modo molto pratico e veloce. Molto articolata è anche la componentistica della toolbar relativa ai form. Da notare una comoda scorciatoia (il **jump menu**) che combina una combo box con del codice Java script che passa a pagine diverse a seconda della scelta sul controllo: è un costrutto relativamente semplice, ma è pur sempre comodo trovarselo già fatto.

In tutte le finestre che operano sui tag è presente un testo cliccabile dal titolo **Tag Info**: cliccandoci sopra si apre il manuale di riferimento sul tag in questione, con una ottima sintesi delle principali caratteristiche sintattiche e funzionali del tag stesso.

Il manuale da cui sono tratte le informazioni è della O'Reilly, e non è l'unico in dotazione con Dreamweaver: nella palette **Reference** trovate della documentazione interessante su CSS, JavaScript e tecnologie server-side che potete consultare comodamente mentre lavorate sui vostri siti. Oltre alle informazioni di riferimento da consultare, per lavorare con i tag Dreamweaver vi mette a disposizione due palette specializzate: quella delle **Properties**, che cambia a seconda del tag su cui posizionate il cursore e che contiene dei controlli per gestire gli attributi più significativi, e quella del **Tag Inspector**, che oltre a mostrare una visualizzazione ad albero del vostro documento vi mette a disposizione una lista meno intuitiva degli attributi, ma completa. Nella finestra delle proprietà troviamo inoltre due interessanti caratteristiche: la possibilità di gestire il tag sia tramite attributi HTML che attraverso gli stili CSS, e la possibilità di creare link o riferimenti a file (quando questi siano richiesti dal tag) tramite il drag & drop sull'icona del **Point to File**.

La **Figura 6** vi mostra il pulsante di switch tra HTML e CSS e il **Point to File** su cui trascinare i file dalla palette **Files**, insieme alla finestra di ispezione dei tag.

Ovviamente le palette sono sincronizzate con la finestra di editazione visuale e del codice, per cui selezionando un elemento dall'albero dei tag viene selezionata la parte corrispondente nell'area di lavoro, così come avviene con la barra contestuale di navigazione del documento posta al fondo della finestra di editing.

RIFERIMENTI

Oltre alle informazioni di riferimento da consultare, per lavorare con i tag Dreamweaver vi mette a disposizione due palette specializzate: quella delle Properties, che cambia a seconda del tag su cui posizionate il cursore e che contiene dei controlli per gestire gli attributi più significativi, e quella del Tag Inspector, che oltre a mostrare una visualizzazione ad albero del vostro documento vi mette a disposizione una lista meno intuitiva degli attributi, ma completa.

L'AIUTO OFFERTO DAL PRODOTTO

Nella fase di preparazione di una pagina web può essere utile a volte riutilizzare delle parti di codice che costituiscono degli aspetti consistenti del nostro sito: a questo fine, Dreamweaver vi offre una caratteristica che vi permette di creare dei blocchi di HTML richiamabili a comando.

Si tratta degli snippet, per i quali esiste una palette a parte e che rappresentano frammenti di codice che si possono inserire nella pagina cliccando due volte su di essi nella palette. Sono di due tipi:

1) **wrap**, inseribili prima e dopo la selezione corrente sul testo del codice.

2) **block**, un unico blocco di HTML da incastrare nel documento.

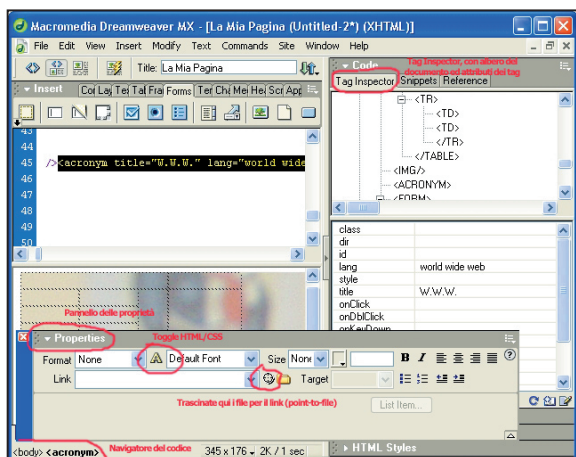



Fig. 6 - tag inspector, properties e code navigator.

La gestione degli **snippet** (creazione, catalogazione, etc) avviene tutta all'interno della palette relativa, mentre – come dicevo sopra – per inserire uno snippet nel documento bisogna fare doppio click sullo snippet sempre sulla palette(**Fig. 7**).

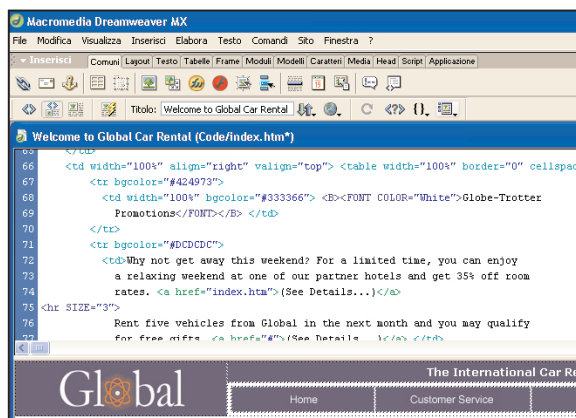



Fig. 7 - Layout e codice sempre sottocontrollo, e con gli snippet pronti all'uso!

Infine, per concludere questa parte sulle caratteristiche di base del prodotto, non possiamo non accennare ai controlli che Dreamweaver vi permette di effettuare sulle pagine che scrivete, sia che siano state composte automaticamente che codificate manualmente.
Una serie di controlli di validità del codice HTML dal punto di vista formale vi consente di essere sicuri che anche i browser più rigidi quali Netscape accettino la pagina dal punto di vista dell'HTML e la possano visualizzare correttamente.
Per controllare quello che è stato scritto selezionate **File / Check Page / Validate Markup...** per validare il codice come HTML, oppure **File / Check Page / Validate as XML...** se utilizzate XHTML.
Inoltre potete richiedere una validazione specifica a fronte di un dato browser tra quelli supportati da Dreamweaver: selezionando **File / Check Page / Check Target Browsers...** (**Fig. 8**) vi viene richiesto con quale browser volete verificare la compatibilità del vostro documento.

Fig. 8 - Ecco il risultato del test di compatibilità del nostro codice con l'Explorer 5.0 di Microsoft

In tutti i casi sopraelencati, alla fine della validazione si aprirà una finestra di palette adeguata con i risultati dell'operazione, da cui potrete prendere spunto per eventuali correzioni al vostro codice.
Un altro check-up interessante offerto da Dreamweaver è quello dell'accessibility.
Questo termine sta ad indicare la possibilità che la pagina venga utilizzata con successo su browser per portatori di handicap (ad esempio, per i non vedenti) e che contenga quindi gli attributi e i tag necessari per fornire informazioni supplementari utili alle persone portatrici di handicap (ad esempio, le immagini devono contenere l'attributo **alt** affinché i browser per ciechi possano "leggere" la descrizione della figura). Tale controllo viene richiesto tramite **File / Check Page / Check Accessibility** e restituisce una lista di tutte le incongruenze rispetto alle esigenze dei navigatori HTML che rispondono ad esigenze particolari.

COSA BOLLE IN PENTOLA

Per oggi abbiamo dato una veloce occhiata alle caratteristiche di base di Dreamweaver e agli aiuti che offre per creare una pagina web. Successivamente parleremo delle capacità di questo meraviglioso software di gestire interi siti e verificare la validità dei link tra le varie pagine. Parleremo anche dei template e di come questi semplifichino di molto il compito di creare siti con un look consistente.
Tra le funzioni più avanzate ci sono i vari wizard per la creazione di pagine dinamiche lato server, con molto codice già scritto e da customizzare: parleremo anche di questo, concentrandoci sempre su quello che si può fare in maniera automatica con Dreamweaver, per scoprire come può esserci di aiuto per essere più produttivi.
A questo punto quindi non mi resta che salutarvi, e darvi appuntamento alla prossima puntata!

Federico Mestrone

Come creare una connessione locale per scambiare dati tra due filmati Flash

Una delle più interessanti novità introdotte da Flash MX è l'oggetto LocalConnection, che permette di inviare e ricevere dati tra filmati distinti, che consente anche di mettere in comunicazione filmati eseguiti in applicazioni differenti.

SUL CD
\soft\codice\local.zip

Durante lo sviluppo di un progetto sul web, l'importanza di una tecnica di comunicazione tra diversi filmati può essere rilevante. Con la versione 5 di Flash, per ottenere lo scambio di dati tra due oggetti eseguiti sullo stesso client era necessario ricorrere agli **FScommand** oppure a JavaScript, ottenendo risultati non sempre apprezzabili, soprattutto quando si incappava nei soliti problemi di incompatibilità tra browser diversi. Ipotizziamo ad esempio di dover sviluppare un sito in HTML nel quale solo una determinata sezione è contraddistinta da un banner realizzato con Flash MX, e supponiamo di voler controllare questo banner con dei pulsanti posti in un'altra parte del sito: grazie al nuovo oggetto **LocalConnection** oggi un'operazione simile è davvero semplice. In pratica, basta inserire alcune semplici righe di codice ActionScript sia nel filmato che invia i dati sia nel filmato che li riceve. I dati in questione possono essere di qualsiasi tipo: coordinate che gestiscono il movimento di clip filmati, valori booleani, stringhe di testo e così via. Basterà decidere in base a quale evento generato dall'utente deve avvenire l'invio dei dati (ad esempio la pressione di un tasto il movimento del mouse ecc...) per impostare una connessione locale tra i due filmati, che dialogheranno come se facessero parte di un solo oggetto. Un esempio on line è disponibile all'indirizzo **www.dynamicdesign.it /local**, mentre nella cartella **\soft\codice\local.zip** del cd-rom allegato alla rivista sono stati messi a disposizione sia i file sorgenti completi che una versione incompleta che useremo come punto di partenza nel corso di questo tutorial. Il risultato finale del nostro lavoro saranno due filmati flash denominati **invio.fla** e **ricevi.fla**, che poi esporteremo e posizioneremo in una stessa pagina denominata **index.html**. Apriamo quindi il file **invio_incompleto.fla** che si trova sul cd-rom e salviamolo come **invio.fla** in una cartella del nostro hard disk. Questo file è dotato nella sua libreria di tutti i pulsanti, i clip e gli elementi grafici necessari alla realizzazione del nostro tutorial: non ci dilungheremo sullo sviluppo della parte grafica per poterci dedicare con maggiore attenzione al codice che aggiungeremo al suo interno (**Figura 1**). Creiamo tre nuovi livelli premendo il tasto **Inserisci livello** e chiamiamoli rispettivamente, dal basso verso l'alto: **pulsanti, grafica** e **azioni.** Nel livello **pulsanti** inseriamo quindi delle istanze dei pulsanti **on** e **off**, prelevandoli dalla libreria e posizionandoli in basso a destra, uno accanto all'altro. Nella parte in alto a sinistra dello stage, invece, inseriamo i tre pulsanti **colore1**, **colore2** e **normale**, disponendoli uno sopra l'altro. Preleviamo l'unico clip della libreria e, dopo avergli dato come nome di istanza **segno** (nel pannello **Proprietà**), posizioniamolo accanto al pulsante **maschera off**. Infine, nel livello grafica, inseriamo i due simboli grafici **titolo** e **fascia**. A questo punto possiamo associare il codice ai nostri pulsanti. Selezioniamo il pulsante **off**, apriamo il pannello **Azioni** (**Finestra> Azioni**) e inseriamo il seguente codice:

```
1 on(release){
2 valore="acceso";
3 _root.segno._x=10;
4 }
```

Al rilascio del pulsante viene passata alla variabile **valore** la stringa **"acceso"**, mentre il clip **segno** viene posizionato accanto al pulsante **on**. Discorso analogo per il pulsante **on**, che però avrà un codice del tipo:

```
1 on(release){
2 valore="spento";
3 _root.segno._x=114;
4 }
```

Fig. 1- Nella libreria si possono vedere un simbolo grafico, un clip filmato e 5 pulsanti già pronti per l'uso.

Fig. 2 - Nel livello pulsanti sono posizionati cinque pulsanti e un clip, necessari per attivare gli eventi che condizioneranno l'altro filmato flash.

Per il pulsante **colore1** il discorso è ancora più semplice, in quanto il solo valore messo in gioco è la variabile **colore**. Inseriamo un codice identico a quello di **colore1** (mostrato nella **Figura 3**) nei pulsanti **colore2** e **normale**, cambiando il valore attribuito alla variabile **colore**, che sarà rispettivamente: **"due"** e **"tre"**.

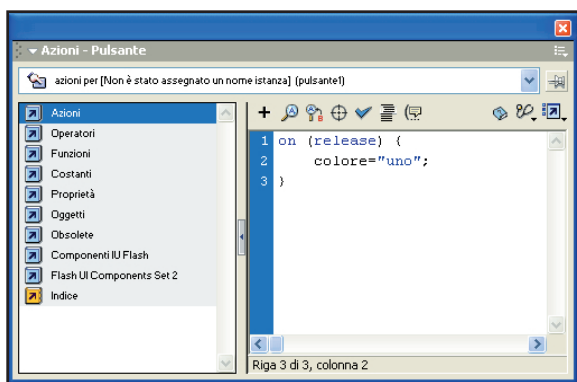



Fig. 3 - In questo caso, cliccando il tasto viene passata alla variabile colore la stringa "uno".

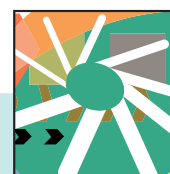
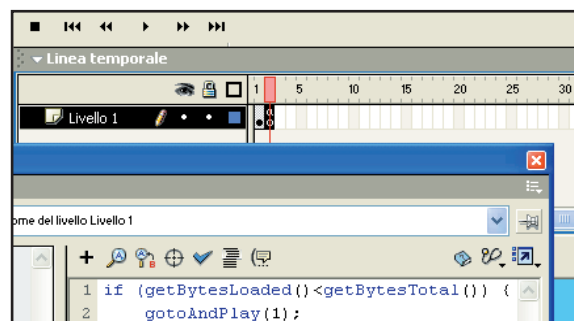
A questo punto possiamo inserire il codice nel fotogramma **azioni**:

```
1 _root.onMouseMove = function() {
2 invia = new LocalConnection();
3 invia.send("tutorial", "metodo", _root._xmouse,
                    _root._ymouse,_root.valore,_root.colore);
4 invia.close();
5 }
```

Per quanto riguarda **_root.onMouseMove=function() {istruzioni}**, rientriamo nella prassi delle "tradizionali" istruzioni **ActionScript**, che riguardano la gestione di un singolo filmato. Si tratta cioè del gestore di evento **onMouseMove**, che, attraverso questa sintassi, introdotta a partire dall'ultima versione di Flash, consente di definire una funzione che verrà eseguita al verificarsi dell'evento.
In altre parole, le istruzioni comprese tra le parentesi graffe saranno eseguite ogni volta che l'utente muoverà il mouse.
Nel nostro caso, però, le istruzioni non coinvolgono questo filmato, ma un altro che non abbiamo ancora definito. Nella riga numero 2, viene creata un'istanza dell'oggetto **LocalConnection** denominata **invia**. Successivamente (riga 3), a questa istanza viene applicato il metodo **send()**, la cui sintassi generale può essere così schematizzata:

```
nomeistanza.send(nomeconnesione,nomemetodo, parametro_1,
                                    parametro_2, … , parametro_n)
```

Questo metodo definisce: un nome univoco che sarà usato durante la connessione locale, il nome di un metodo creato appositamente per la connessione locale ed una serie di valori da inviare allo stesso filmato ricevente. Nel nostro codice abbiamo indicato come nome della connessione la stringa **"tutorial"**, come nome del metodo il termine generico **"metodo"** e, come parametri, la coordinata **x** del mouse è la coordinata **y** del mouse. Infine abbiamo specificato l'invio delle variabili **valore** e **colore**. Mentre le coordinate del mouse vengono determinate dal movimento operato dall'utente, il valore delle variabili è definito dai pulsanti messi a disposizione nel filmato. Per cui, se l'utente preme ad esempio il pulsante **colore2**, a questo sarà attribuita la stringa **"due"**. Per finire, non ci resta che commentare il metodo **close()**, da utilizzarsi quando si vuole che l'oggetto non accetti più istruzioni: è un procedimento necessario per poter applicare il metodo **connect()** che usa lo stesso parametro **nomeconnesione** nel nostro filmato ricevente. In un certo senso è come se definisse la quantità di dati da inviare al termine dell'evento (nel nostro caso, il movimento del mouse) che gestisce la connessione locale. Una volta ultimato il filmato **invia.fla**, salviamolo e procediamo alla pubblicazione con **File>Pubblica:** nella cartella dove si trova il file sorgente verranno generati i file **invia.html** ed **invia.swf.** A questo punto possiamo procedere alla realizzazione del secondo filmato: preleviamo dal cd-rom **ricevi_incompleto.fla** e salviamolo come **ricevi.fla** nella stessa cartella in cui abbiamo posizionato il file precedente. Anche in questo caso sono già disponibili nella libreria gli elementi grafici necessari. Questo file, a differenza del precedente, poiché contiene una foto prelevata da una raccolta di **Maxi Photo Collection** (la rivista che la nostra casa editrice dedica alla fotografia e alla grafica digitale), peserà qualche kb in più. Per questo motivo è necessario aggiungere un semplice precaricamento. Attiviamo quindi il pannello **Scena** (**Finestra>Scena**) e selezioniamo la scena **precaricamento**: così facendo ci posizioneremo in una scena già creata in precedenza e disposta prima della scena principale. Apriamo la libreria e preleviamo il simbolo grafico **scritta**, per poi posizionarlo al centro dello stage con il pannello **Allinea (Finestra>Allinea)**. Selezioniamo il fotogramma successivo e trasformiamolo in

Fig. 4 - L'istruzione condizionale if rimanda indietro la testina al fotogramma 2, fino a quando non sono stati scaricati sul client dell'utente tutti i kb del filmato.

fotogramma chiave vuoto (**Inserisci>Fotogramma chiave vuoto**). Infine inseriamo una semplice istruzione condizionale che impedisca alla testina di andare avanti fino a quando non sia stato scaricato tutto il filmato **ricevi.swf**. Attraverso il tasto **Modifica scena** o tramite il pannello **Scena** torniamo sulla scena principale e creiamo tre livelli, che dal basso verso l'alto chiameremo rispettivamente: **sfondo**, **clip** e **azioni**.

Apriamo la libreria e posizioniamo al centro dello stage nel primo livello il clip **sfondo**, ricordandoci di dargli come nome **istanza sfondo**; selezioniamo poi il secondo livello e, dopo avere prelevato dalla libreria **clip1**, posizioniamolo al centro dello stage. Infine, con il pannello **Proprietà**, diamo al clip il nome **istanza clip1**. Dopo aver posizionato i clip che compongono la parte grafica del filmato, possiamo dedicarci al codice, che inseriremo nel primo fotogramma del livello **azioni**.

Prima di tutto definiamo la porzione di codice non riferita alla connessione locale:

```
1 stop();
2 clip1._alpha=50;
3 _root.onEnterFrame=function(){
4 _root.clip1._rotation+=8;
5 }
```

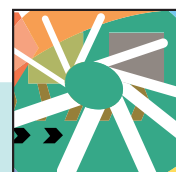
Con queste prime righe non facciamo altro che dare uno stop, in modo da impedire di riavviare nuovamente il precaricamento; inoltre, attribuiamo alla proprietà **_alpha** del **clip1** una trasparenza pari a 50 e, attraverso l'evento **onEnterFrame**, incrementiamo ad ogni lettura di fotogramma la proprietà **_rotation** del clip **filmato clip1** (che quindi ruoterà sul suo asse continuamente). A partire dalla riga seguente (vedere riga 6 più avanti) viene invece stabilita una connessione locale che riceve i dati inviati dal filmato **invia.swf**. Lo schema generale del codice da inserire nel filmato ricevente è in genere del tipo:

```
nomeistanzaricevente = new LocalConnection();
nomeistanzaricevente.nomemetodo = function
                    (parametro_1,parametro_2, ... , parametro_n) {
//istruzioni che utilizzano i parametri indicati
}
nomeistanzaricevente.connect("nomeconnesione");
```

In pratica, dopo aver istanziato un nuovo oggetto **LocalConnection**, viene applicato a quest'ultimo il metodo creato nel filmato mittente, cioè nel filmato che invia i dati. I parametri indicati sono corrispondenti a quelli che vengono specificati nel metodo **send()** nel filmato mittente. Anche se questi parametri si chiamano diversamente, viene comunque creata una corrispondenza in base all'ordine in cui sono disposti: quindi se, ad esempio, con il metodo **send()** abbiamo indicato la stringa **"prova"** come valore del primo parametro, questo valore sarà passato al primo parametro di **function()** nel filmato destinatario (il filmato che riceve i dati). Poiché quello che conta è la corrispondenza a livello di posizione dei due parametri, la comunicazione tra i due filmati avviene anche se i due parametri si chiamano diversamente. Il metodo **connect()** prepara un oggetto **LocalConnection** a ricevere istruzioni da un comando **LocalConnection.send()**, ed ha come parametro il nome della connessione specificato nel filmato mittente. Analizziamo quindi la parte il codice inserito nel filmato **ricevi.fla** a partire dalla sesta riga:

```
6 ricevi = new LocalConnection();
7 ricevi.metodo = function(var1, var2, var3,var4) {
8 clip1._x = var1;
9 clip1._y = var2;
10 if (var3 == "acceso") {
11 sfondo.setMask(clip1);
12 }
13 else if (var3 == "spento") {
14 sfondo.setMask(null);
15 }
```

Alla proprietà **_x** e **_y** del **clip1** vengono passate le coordinate del mouse che l'utente determina muovendosi sul filmato mittente, per cui il movimento effettuato sul filmato **invia** avrà un effetto sul filmato **ricevi**. Oltre ad applicare il principio generale, attraverso delle semplici **if**, decidiamo l'istruzione da eseguire in base al valore di **var3**, che a sua volta corrisponde al parametro del filmato mittente **_root.valore** e viene deciso in base alla scelta operata dall'utente. Le istruzioni condizionali prevedono due possibilità: alla variabile viene assegnata una stringa il cui valore è "acceso" oppure "spento". Nel primo caso, viene applicato il metodo, (disponibile a partire da Flash MX), che consente di usare come maschere i clip filmati - la cui funzione di maschere può essere attivata o disattivata in fase di esecuzione. La sintassi davvero molto semplice è del tipo: **Clip_da_mascherare.setMask (clip_maschera)**; nella riga 11 del nostro listato specifichiamo che intendiamo applicare una maschera al clip **sfondo** e che vogliamo che tale maschera sia costituita da **clip1**. Per disattivare la modalità maschera assunta dal clip, basta applicare lo stesso metodo specificando come parametro **null**. A partire dalla riga 16 c'è una sequenza di tre istruzioni condizionali che invece impostano il colore del clip **sfondo** in base al valore di **var4**, che raccoglie i dati inviati dalla variabile colore del filmato **invia**.

```
16 if(var4=="due"){
17 nuovo=new Object();
18 nuovo.ra=-100;
19 nuovo.rb=0;
20 nuovo.ga=100;
21 nuovo.gb=4;
22 nuovo.ba=52;
23 nuovo.bb=-19;
24 nuovo.aa=100;
25 nuovo.ab=0;
26 miocolore=new Color("sfondo");
27 miocolore.setTransform(nuovo);
28 }
```

Questo procedimento costituisce la tecnica utilizzata normalmente per impostare attraverso ActionScript la colorazione, che è possibile applicare ai simboli sfruttando le opzioni disponibili nel pannello delle proprietà scegliendo tra le opzioni del menu **colore avanzato** e premendo successivamente il tasto **impostazioni**. Se la condizione è verificata, viene creato un oggetto generico in cui sono impostate otto proprietà che corrispondono alle otto opzioni messe a disposizione nel pannello **Effetto**. Infine viene creata un'istanza dell'oggetto **Color** (riga 26) e viene applicato il metodo **setTransform()**, che ha come parametro l'oggetto appena creato. Questo tipo di colorazione, a differenza di quella semplice, che utilizza il metodo **setRGB()**, è particolarmente indicata per i clip caratterizzati (come nel nostro caso) da una bitmap.
Le rimanenti istruzioni condizionali che contengono comandi analoghi a quelli appena analizzati, impostano, in base al valore assunto della variabile **var4**, altre due colorazioni differenti, una con sfumatura verde e un'altra che ripristina i valori "di default" della foto.

```
29 if(var4=="due"){
30 nuovo=new Object();
31 nuovo.ra=-100;
32 nuovo.rb=0;
33 nuovo.ga=100;
34 nuovo.gb=4;
35 nuovo.ba=52;
36 nuovo.bb=-19;
37 nuovo.aa=100;
38 nuovo.ab=0;
39 miocolore=new Color("sfondo");
40 miocolore.setTransform(nuovo);
41 }
42 if(var4=="tre"){
43 nuovo=new Object();
44 nuovo.ra=100;
45 nuovo.rb=0;
46 nuovo.ga=100;
47 nuovo.gb=0;
48 nuovo.ba=100;
49 nuovo.bb=0;
50 nuovo.aa=100;
51 nuovo.ab=0;
52 miocolore=new Color("sfondo");
53 miocolore.setTransform(nuovo);
54 }
55 }
```

Una volta definite le istruzioni da eseguire usando le variabili ricevute, applichiamo il metodo **connection()**, che restituisce un valore booleano **true**, se nessun altro processo in esecuzione sullo stesso client ha già applicato questo comando utilizzando lo stesso valore per **nomeconnessione** mentre, in caso contrario, restituisce **false**. Il metodo **connection()** si applica all'istanza del filmato destinatario ed ha come unico parametro il nome di connessione definito nel filmato mittente. Nel nostro caso sarà:

```
56 ricevi.connect("tutorial");
```

Ultimato anche il filmato **ricevi.fla**, non ci resta che procedere alla pubblicazione con **File>Pubblica**.

```
index_incompleto - Blocco note
File  Modifica  Formato  Visualizza  ?
</head>
<body>
<table border="0" cellspacing="3" align="center">
<tr><td colspan="2" style="font-family: verdana; font-size:14pt; color:#990000">
<tr>
<td>
<table border="0" cellspacing="2">
<tr><td>
<img src="vuoto.gif" width="300" height="150" border="1">
</td></tr>
<tr><td>
<span class="didas">oggetto che invia i dati</span>
</td></tr>
</table>
```

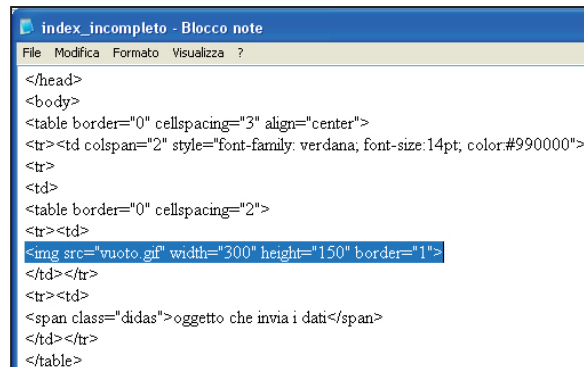
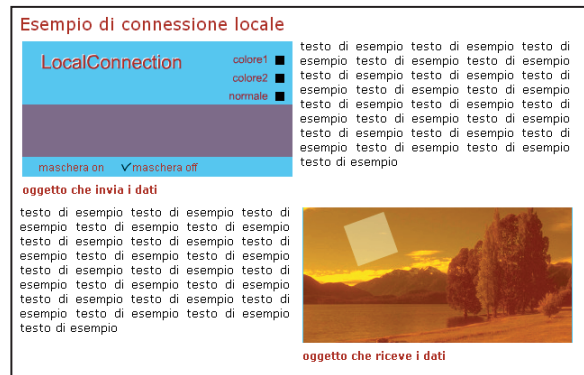
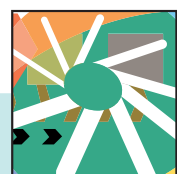
Fig. 5 - Il primo dei due tag ←img→ presenti nella tabella html.

Sempre nel cd-rom allegato alla rivista, è stato fornito il file **index_incompleto.html**: preleviamolo e posizioniamolo nella cartella in cui si trovano i due file **swf** e i relativi file **html** creati durante la pubblicazione. Se osserviamo il sorgente, possiamo notare che questo file html è costituito da una tabella che ingloba del testo di esempio e una gif, richiamata in due differenti celle, la cui unica funzione è segnalare i punti vanno inseriti i due filmati **swf**.
Apriamo quindi il file **invia.html** e, dopo aver selezionato la porzione di codice che va da **<object>** ad **</object>** copiamola. Successivamente, apriamo il file **index_incompleto.html** e incolliamo quanto copiato al posto del primo dei due tag **<img>**.

Fig. 6 - Il filmato in basso a destra viene controllato in base ai movimenti operati sul primo in alto a sinistra.

Ripetiamo la stessa operazione copiando il tag **<object></object>** del file ricevi.html e incollandolo al posto del secondo tag **<img>** del file **index_incompleto.html**.
A questo punto è possibile rinominare il file, chiamandolo **index.html** e lanciarlo per un'anteprima.

Maurizio Battista

PHP-Nuke 6.0 Web portal system (parte seconda)

Nello scorso appuntamento abbiamo analizzato le caratteristiche di PHP-Nuke, il più conosciuto CMS (Content Management System) Open Source. Questo mese vedremo come configurare il sistema e come personalizzarne la grafica.

SUL CD
\soft\codice\Sorgente_webmatrix1.zip

Il mese scorso abbiamo visto quali siano l'utilità e le funzioni di un generico sistema per la gestione dei contenuti (CMS), con particolare riferimento a PHP-Nuke (**http://phpnuke.org**), il popolare sistema per lo sviluppo di portali, interamente scritto in PHP. Inoltre abbiamo esaminato i passi necessari per installare PHP-Nuke sia in locale, sul nostro PC (o server) di casa, che in remoto su uno spazio web gratuito, più precisamente quello messo a disposizione da Lycos (**http://www.tripod.lycos.co.uk/myaccount /freehosting**). Completata con successo l'installazione possiamo affrontare la fase di configurazione, in modo da personalizzare il nostro nuovo portale. Per prima cosa entriamo nella pagina di amministrazione che troviamo, nel caso di installazione in locale, all'URL **http://localhost/admin.php**. Dopo aver specificato i dati dell'amministratore (ID e password) ci verrà presentata la schermata di amministrazione, che contiene tutte le icone che consentono di accedere alle diverse sezioni e funzionalità del sito. I primi parametri che dovremo personalizzare saranno quelli relativi alla sezione **Preferences**. All'interno di questa sezione troveremo diverse voci, ma quelle più importanti per il corretto funzionamento del portale sono poste nel primo riquadro (**General Site Info**), più in particolare andranno specificate le voci:

Title	Custom Title	Status	Visible to	Functions
Addon_Sample	Addon Sample	Inactive	Administrators Only	[Edit | Activate | Put in Home]
AvantGo	AvantGo	Inactive	All Visitors	[Edit | Activate | Put in Home]
Content	Content	Inactive	All Visitors	[Edit | Activate | Put in Home]
Downloads	Downloads	Active	All Visitors	[Edit | Deactivate | Put in Home]
Encyclopedia	Encyclopedia	Inactive	All Visitors	[Edit | Activate | Put in Home]
FAQ	FAQ	Inactive	All Visitors	[Edit | Activate | Put in Home]
Feedback	Feedback	Inactive	All Visitors	[Edit | Activate | Put in Home]
Forums	Forums	Inactive	All Visitors	[Edit | Activate | Put in Home]
Journal	Journal	Inactive	All Visitors	[Edit | Activate | Put in Home]
Members_List	Members List	Inactive	Registered Users Only	[Edit | Activate | Put in Home]
News	News	Active (In Home)	All Visitors	[Edit | Deactivate | Put in Home]
Private_Messages	Private Messages	Inactive	All Visitors	[Edit | Activate | Put in Home]
Recommend_Us	Recommend Us	Inactive	All Visitors	[Edit | Activate | Put in Home]
Reviews	Reviews	Inactive	All Visitors	[Edit | Activate | Put in Home]
Search	Search	Inactive	All Visitors	[Edit | Activate | Put in Home]

Fig. 1 - I moduli a disposizione di PHP Nuke.

- **Site Name** il nome del sito, utilizzato sia come titolo per le pagine del portale, sia come contenunto di alcuni meta tag;
- **Site Slogan** utile per indicare una "descrizione" del contenuto del sito, infatti questa informazione sarà utilizzata da PHP-Nuke per comporre il meta tag "**description**";
- **Site Start Date** data di pubblicazione del sito; informazione utilizzata dal modulo **Statistics** (statistiche di accesso al sito);
- **Administrator Email** email dell'amministratore del sito, al quale saranno inviate le comunicazioni di servizio e quelle inoltrate dagli utenti tramite il modulo **Feedback**;
- **Allow Anonymous to Post?** tramite questa voce potremo specificare se permettere agli utenti anonimi, cioè quelli che non hanno un account sul nostro sito, di scrivere commenti;
- **Default Theme for your site** permette di selezionare l'aspetto grafico (tema) del sito. PHP-Nuke 6.0 dispone di diversi temi già pronti, ma molti altri possono essere scaricati dalla rete;

Queste rappresentano solo le impostazioni di base, cioè il minimo insieme di informazioni da fornire al sistema per poter funzionare. L'area **Preferences** permette di impostare molte altre caratteristiche in modo da poter realizzare il portale più adatto alle nostre esigenze.

FUNZIONALITÀ DEL PORTALE

A questo punto siamo pronti ad impostare il nostro portale in modo che possa contenere e gestire solo i servizi e le informazioni che ci intressano. Supponiamo di voler realizzare un sito dotato di 4 funzionalità di base, in modo da gestire: notizie, download, sondaggi e link. Gli strumenti messi a disposizione da PHP-Nuke per gestire queste informazioni sono i moduli, componenti che saranno mostrati nella parte centrale delle pagine del portale. Quindi, sempre dalla sezione di amministrazione, clicchiamo sull'icona **Modules** e configuriamo i moduli in base alle nostre esigenze. La tabella che ci verrà mostrata (**Fig. 1**) raccoglie tutti i moduli di cui dispone PHP-Nuke, insieme ai link tramite i quali modificarli. In particolare, per ogni modulo, saranno mostrate le seguenti informazioni:

- Title indica il titolo del modulo; corrisponde al nome della directory (relativa alla **/modules**) nella quale si trova il codice PHP del modulo;

- **Custom Title** indica il nome con il quale il modulo sarà visualizzato sul portale;
- **Status** indica lo stato del modulo: attivo, inattivo, oppure attivo in homepage;
- **Visible to** indica gli utenti abilitati alla visualizzazione del modulo: tutti gli utenti, solo gli amministratori, oppure solo gli utenti registrati;
- **Functions** contiene i link alle funzioni di amministrazione del modulo: modifica, attivare/disattivare, visualizzare in homepage;

A questo punto possiamo modificare le impostazioni dei moduli, tramite i link **Activate/Disactivate** della colonna **Functions**, in modo da rendere attivi solo quelli che serviranno ai nostri scopi, e cioè:

- **News**
- **Downloads**
- **Surveys**
- **Web_Links**

Come possiamo notare, una delle righe relative ai moduli disponibili in PHP-Nuke, solitamente quella delle **News**, è evidenziata in grassetto. Questo sta ad indicare che il modulo al quale corrisponde sarà quello mostrato nell'homepage del nostro portale, per variarlo non dovremo fare altro che intervenire sul link **Put in Home**, in corrispondenza del modulo che intendiamo attivare. La seconda impostazione che determina le funzioni del portale che intendiamo costruire è quella dei blocchi, cioè i box laterali che sono visualizzati su ogni pagina di PHP-Nuke.
Per configurarli basterà accedere all'apposita area di amministrazione tramite il link **Blocks**.

Fig. 2 - La figura è abbastanza simile alla Figura 1; la differenza maggiore consiste nella possibilità di stabilirne la posizione (Position).

Questa pagina (**Fig. 2**) è abbastanza simile a quella vista precedentemente, la differenza maggiore consiste nella possibilità di stabilirne la posizione (**Position**), cioè se visualizzarlo nella colonna di destra o di sinistra, e l'ordine (**Weight**); infatti non dobbiamo dimenticare che i blocchi, a differenza dei moduli, saranno mostrati tutti contemporaneamente.

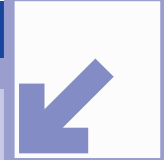
INSERIAMO I CONTENUTI

Dopo aver effettuato la configurazione base del nostro nuovo portale, siamo pronti a definire ed inserire i contenuti. In genere, il modulo più utilizzato dai siti costruiti tramite PHP-Nuke, non a caso è impostato anche come default per l'homepage, è quello delle **News**. Iniziamo con definire quali argomenti (**topic**) saranno affrontati dalle notizie pubblicate sul sito. Quindi entriamo nella sezione **Topics**, eventualmente cancelliamo i topic che non ci interessano ed inseriamo i nuovi. I campi richiesti per l'inserimento di una nuova categoria sono:

- **Topic Name** indica il nome dell'argomento, ma solo per uso interno, va quindi indicato senza spazi e della lunghezza massima di 20 caratteri;
- **Topic Text** indica la descrizione completa (massimo 40 caratteri) del topic;
- **Topic Image** rappresenta l'immagine associata al topic; possiamo aggiungerne di nuove copiandole nella directory **/images/topics/;**

Per la nostra prova inseriremo 3 topic relativi ad altrettante riviste, e precisamente: "Programmare il Web", "ioProgrammo" e "Linux Magazine" (**Fig. 3**).

Fig. 3 - Per la nostra prova abbiamo inserito tre topic relativi ad altrettante riviste, e precisamente: "Programmare il Web", "ioProgrammo" e "Linux Magazine"

Quindi possiamo entrare nella sezione **Add Story** ed iniziare ad inserire le notizie. Per ognuna di esse dovremo indicare le classiche informazioni quali: titolo, argomento, descrizione breve e testo; ma inoltre avremo la possibilità di indicare se attivare o meno i commenti degli utenti, pubblicarla subito oppure in corrispondenza di una data ed ora prefissati, ed infine se allegare un sondaggio alla notizia. Passiamo ora a riempire la sezione dedicata al download; innanzitutto entriamo nell'apposita area di amministrazione tramite l'icona **Downloads** ed iniziamo a definire le categorie in base alle quali suddividere i file. PHP-Nuke ci permette di organizzare il software tramite categorie e sottocategorie; ad esempio potremmo creare la categoria **"Software"** e, al suo interno, creare

le sottocategorie **"Windows"**, **"Linux"**, ecc. Solo dopo questa fase potremo iniziare ad inserire i programmi che intendiamo rendere disponibili ai nostri utenti/visitatori. In pratica per ogni programma dovremo riempire una scheda specificando: il nome, il link da cui scaricarlo, la categoria/sottocategoria a cui appartiene, la descrizione, il nome e l'email dell'autore, la dimensione in byte, la versione, l'homepage ed il numero di hits iniziali, cioè il numero di volte che il file è stato scaricato. Come avrete avuto modo di notare, l'inserimento dei contenuti è un'operazione abbastanza semplice ed agevole da compiere. Infatti, oltre ad avvenire tramite semplici interfacce web, il sistema nella sua complessità è abbastanza lineare e coerente. Pertanto, considerando inoltre che le operazioni da compiere per riempire le altre sezioni del sito sono analoghe a quelle appena vista, non ci dilungheremo oltre lasciandovi il compito per esercizio.

PERSONALIZZARE LA GRAFICA

Probabilmente una delle prime domande che si pone un nuovo utente di PHP-Nuke riguarda la personalizzazione della grafica. Per questi scopi vengono utilizzati i temi (**themes**), alcuni già disponibili nel pacchetto di PHP-Nuke. Se questi non dovessero essere di nostro gradimento, o semplicemente desideriamo avere per il nostro sito un aspetto un po' più originale, potremmo cercare in rete qualcosa di più adatto. Sono infatti decine i siti che offrono temi per PHP-Nuke, sia gratuitamente che a pagamento. In genere vengono distribuiti in un file compresso (**.zip, .tgz, .ecc**) che, una volta scaricato, dovrà essere decompresso nella directory **/themes/**; quindi dovremo accedere all'area di amministrazione, scegliere l'icona Preferences ed indicare, alla voce **Default Theme for your site** il nome del tema che abbiamo appena installato.
Naturalmente, per creare veramente qualcosa di originale, la cosa migliore da fare sarebbe quella di creare un proprio tema. L'operazione può sembrare complessa, ma con una minima conoscenza del linguaggio PHP, ed ovviamente dell'HTML, e magari iniziando a modificare un tema già pronto, non sarà difficile ottenere risultati soddisfacenti.
Come abbiamo già visto i temi si trovano nella directory **/themes/**, analizzandone il contenuto notiamo però un piccolo problema, cioè la diversa organizzazione dei file. Infatti, a parte le directory standard **images** e **style** che contengono rispettivamente le immagini (icone ed altro) ed il foglio di stile utilizzati dal tema, notiamo 2 diverse implementazioni.
La prima, ad esempio quella utilizzata da **DeepBlue** (tema di default), è realizzata in pratica tramite un unico file, **theme.php**, che include al suo interno sia il codice PHP necessario al sistema, sia il codice HTML relativo alla presentazione grafica del tema.
La seconda modalità utilizzata dai temi, ad esempio utilizzata da 3D-Fantasy, prevede invece una gestione più modulare, tramite l'utilizzo di diversi file HTML/PHP che vengono richiamati (inclusi) dal **theme.php**:

- **blocks.html** per la gestione dell'aspetto grafico dei blocchi laterali;
- **center_right.html** per la gestione dello spazio che intercorre tra la zona centrale dalla colonna destra;
- **footer.html** per la gestione dell'aspetto grafico del footer, ossia il riquadro a fondo pagina;
- **header.html** per la gestione dell'aspetto grafico dell'header, ossia l'intestazione della pagina (logo, banner, ecc.);
- **left_center.html** per la gestione dello spazio che intercorre tra la zona centrale dalla colonna sinistra;
- **story_home.html** per la gestione della visualizzazione delle notizie;
- **story_page.html** per la gestione della visualizzazione delle notizie nel formato completo;
- **tables.php** contiene le funzioni per definire l'HTML relativo all'apertura e chiusura delle tabelle: la struttura più utilizzata da PHP-Nuke;

Questa organizzazione rende il tema più semplice da gestire e modificare, a differenza del metodo che utilizza il solo file **theme.php**, nel quale il codice HTML da modificare e personalizzare andrà ricercato all'interno delle funzioni di cui si compone. Con un po' di pratica diventeremo abilissimi nel modificare i temi esistenti oppure crearne di nuovi; infatti, come spesso avviene in questo campo, le cose sono più difficili a dirsi che a farsi!

Fig. 4 - Un esempio di un semplice tema ottenuto modificando il DeepBlue

CONCLUSIONI

Questo articolo concludiamo l'argomento PHP-Nuke, iniziato nel numero precedente. Abbiamo visto i vantaggi derivanti dall'utilizzo di un CMS come PHP-Nuke, come installarlo e dove intervenire per utilizzarne le funzionalità adatte ai nostri scopi; ora non vi resta altro da fare che iniziare a costruire il vostro portale.

Antonio Pasqua

Un Guestbook XML per il tuo sito

Il Guestbook è una delle funzionalità maggiormente diffuse nei siti web. In questo articolo scopriremo come implementarne uno, sfruttando la semplicità di ADO.NET nella gestione di informazioni relazionali con file XML.

SUL CD
\soft\codice\codice_guestbook.zip

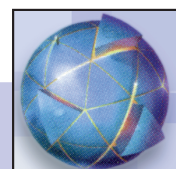
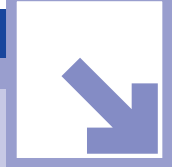
Il libro degli ospiti (guestbook) è uno strumento molto utilizzato dai webmaster desiderosi di offrire ai visitatori la possibilità di lasciare un "segno del proprio passaggio". Tra le varie risorse reperibili in Internet ci sono guestbook di ogni tipo e potenzialità, ma la maggior parte di essi necessita di SQL Server (o Access) per memorizzare i dati relativi ai visitatori. Non tutte le soluzioni di hosting sono predisposte per l'utilizzo di database; per questo motivo vedremo come creare un guestbook rapido e "portabile", indipendente da una particolare scelta di data-storing. La risposta alle nostre esigenze è data dall'accoppiata ADO.NET e XML (Extensible Markup Language). Xml permette di memorizzare i dati relativi ai visitatori, con un approccio alternativo, totalmente text-based, evitando l'onere di connessioni ODBC a server con driver appropriati. Microsoft ha compreso fino in fondo l'importanza di xml nel futuro del web, infatti in .NET ci sono tutti gli strumenti necessari per la creazione di soluzioni basate su questo standard. Più nel dettaglio le classi adibite alla manipolazione di file xml sono racchiuse in due principali namespace: **System.Xml** e **System.Data**. Generalmente **System.Xml** è più versatile, ma richiede un lavoro maggiore da parte del programmatore anche per implementare funzionalità piuttosto semplici. **System.Data** è meno flessibile per quel che riguarda Xml, ma favorisce la programmazione, rendendo la stesura di codice più immediata e semplice se le operazioni (come nel nostro caso) si limitano alla scrittura e lettura di file utilizzati come fonte dati. Utilizzeremo oggetti ADO.NET (disponibili in **System.Data**) che forniscono una via efficiente e semplice per trattare le informazioni come se si stesse lavorando su dei veri e propri database relazionali.

IL GUESTBOOK

L'organizzazione dell'applicazione Web è piuttosto semplice e non richiede una pianificazione particolarmente elaborata. Avremo bisogno di una pagina atta all'inserimento dei messaggi (**aggiungi.aspx**), un'altra pagina che visualizzi i commenti già inseriti dai visisitatori (**visualizza.aspx**) e un file xml come back-end in cui memorizzare tutti i dati (**guestbook.xml**). Dal punto di vista dell'interfaccia utente, **aggiungi.aspx** dovrà contenere un form con dei campi per l'inserimento da parte dell'utente, un pulsante di invio, e un collegamento alla pagina di visualizzazione del guestbook. Quando l'inserimento avverrà correttamente, la stessa pagina dovrà segnalare con un messaggio di ringraziamento l'avvenuta operazione. Dietro le quinte la pagina dovrà creare il file xml, se non esiste, e aggiungere l'entry del messaggio appena inviato nel nostro "archivio". L'utente potrà inserire il proprio nome (**autore**), il proprio e-mail (**email**), l'oggetto del commento (**oggetto**) e il commento vero e proprio (**commento**). Come vedremo tra breve, a questi campi si aggiungeranno altri campi determinati in maniera automatica dalla pagina asp. **Autore** ed **oggetto** sono due campi che renderemo obbligatori, **email** sarà opzionale ma effettueremo un controllo sull'input per determinare se l'indirizzo è formalmente corretto, e **commento** potrà essere inserito dall'utente oppure non inserito, in maniera totalmente arbitraria. Sfruttando le nuove caratteristiche di ASP.NET controllare l'input dell'utente e segnalare eventuali errori risulta davvero immediato. **Visualizza.aspx** dovrà invece leggere i commenti presenti nel file xml e mostrarli in maniera ordinata e opportuna, fornendo un link alla pagina di inserimento dei commenti. Il file **guestbook.xml** verrà generato automaticamente dalla pagina aggiungi, mediante l'utilizzo di ADO.NET. La scelta di operare mediante il namespace **System.Data** ed i suoi oggetti mostra sin da subito i suoi grandi vantaggi: per la realizzazione di un applicazione basata su xml, non bisogna necessariamente conoscere tale linguaggio.
Prima di passare all'analisi del codice di ogni pagina dell'applicazione web, definiamo la struttura della tabella e le proprietà dei campi (o colonne) come se stessimo lavorando su di un comune database:

Nome campo	Tipo di dato
id	Int32
crono	DateTime
autore	String
email	String
oggetto	String
commento	String

Come mostrato nella tabella, il campo "id" è un intero a 32 bit. **Id** è necessario per identificare unicamente i sin-

goli commenti, per cui deve essere anche chiave primaria e contatore (si autoincrementa). "crono" è un campo che contiene la data e l'ora del commento ed è di tipo **System.DateTime**. "id" e "crono" sono le due colonne inserite automaticamente dalla pagina, mentre gli altri 4 campi di tipo stringa sono determinati direttamente dall'utente mediante la compilazione del form nella pagina **aggiungi.aspx.** Gli oggetti necessari alla creazione e manipolazione di questa struttura dati nel file xml sono **DataSet, DataTable, DataRow, DataColumn** e in fine, **DataView** per la visualizzazione.

LA PAGINA AGGIUNGI.ASPX

Utilizzando un editor qualsiasi (ad esempio Web Matrix), dovremo creare un'interfaccia utente simile a quella mostrata in **Fig. 1**.

Fig. 1 - La semplice interfaccia utente della pagina aggiungi.aspx

Si presti attenzione nell'utilizzare esclusivamente caselle di testo di tipo **Web Control (<asp:TextBox id="nomescelto" runat="server"></asp:TextBox>)** e non semplici elementi di input html. Gli id delle quattro textbox sono: **txtAutore, txtEmail, txtOggetto, txtCommento,** mentre l'id del pulsante è **btnInserisci.** Si posizioni inoltre tutto il contenuto del form di invio su di un web control Panel denominato **pnlInvio**. Designata l'interfaccia passiamo all'analisi del codice necessario.
Dovremo ottenere come prime righe del file **aggiungi.aspx**, le seguenti:

```
<%@ Page Language="VB" %>
<%@ import Namespace="System.Data" %>
<%@ import Namespace="System.IO" %>
```

La prima istruzione permette di scegliere il linguaggio Visual Basic (potete sostituirlo con C#, se preferite), mentre le altre due righe effettuano l'import dei namespace **System.Data** e **System.IO**. Il namespace **System.IO** è necessario per i controlli sull'esistenza fisica del file xml sul server. Nella pagina si trovano tre "procedure": **Page_Load, InizializzaFile** e **btnInserisci_Click.** La prima e l'ultima sono delle routine di gestione degli eventi, rispettivamente di caricamento della pagina e di click sul pulsante di invio.
Il codice di **Page_Load** è il seguente:

```
Sub Page_Load(sender As Object, e As EventArgs)
        pnlInizio.Visible = True
        strPath = Server.MapPath("guestbook.xml")
        If Not Page.IsPostBack Then
          If Not File.Exists(strPath) Then
            InizializzaFile()
          End If
        End If
End Sub
```

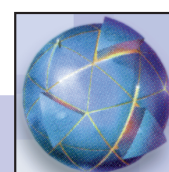
Al caricamento della pagina il pannello contenente tutta l'interfaccia centrale per l'inserimento dei dati (la tabella con le textbox) deve essere resa visibile assegnando il valore **True** alla proprietà **Visible** del **pnlInizio**. Successivamente viene ricercato sul server, il file **guestbook.xml**.
Se tale file non esiste, viene richiamata la procedura **InizializzaFile** che si occupa della creazione fisica del file e di memorizzare la struttura di tabella nello schema XML. Diamo un'occhiata al codice della procedura InizializzaFile:

```
Sub InizializzaFile()
     dsGuestbook = New DataSet()
     Dim dtGuestbook As DataTable =
                                  dsGuestbook.Tables.Add("messaggio")
```

Le precedenti istruzioni creano un'istanza del dataset di nome **dsGuestbook** e instanziano un datatable **dtGuestbook**, aggiungendo quindi una tabella di nome "messaggio" al dataset. Creata la tabella nel dataset, ci occorre definire le 6 colonne (i campi visti in precedenza) che le appartengono. Allo scopo utilizziamo il metodo **Add** di **dtGuestbook.Columns:**

```
Dim dcGuestbook as DataColumn = dtGuestbook.Columns.Add("id",
                                      Type.GetType("System.Int32"))
dtGuestbook.Columns.Add("crono", Type.GetType("System.DateTime"))
dtGuestbook.Columns.Add("autore", Type.GetType("System.String"))
dtGuestbook.Columns.Add("email", Type.GetType("System.String"))
dtGuestbook.Columns.Add("oggetto", Type.GetType("System.String"))
dtGuestbook.Columns.Add("commento", Type.GetType("System.String"))
```

Inzialmente viene istanziato il datacolumn **dcGuestbook** e inserita la colonna id di tipo **int32** all'interno del datatable **dtGuestbook**. Successivamente viene ripetuto l'inserimento di tutte le colonne. Si potrebbe pensare che la creazione dell'istanza **dcGuestbook** sia superflua, visto che l'inseri-

mento delle altre colonne all'interno del datatable è avvenuto senza l'ausilio di alcuna istanza di DataColumn, semplicemente mediante il metodo **Add.** In realtà **dcGuestbook** è necessario perché su di esso dobbiamo effettuare delle operazioni per rendere la colonna **id** nella tabella, di tipo contatore e chiave primaria. Seguono le istruzioni necessarie per definire l'autoincremento:

```
dcGuestbook.AutoIncrement = true
dcGuestbook.AutoIncrementSeed = 1
dcGuestbook.AutoIncrementStep = 1
```

Il DataColumn **dcGuestbook** (corrispondente alla colonna id del datatable) viene reso auto-incrementabile (**AutoIncrement**), a partire da 1 (**AutoIncrementSeed**) con un incremento di 1 (**AutoIncrementStep**) per ogni commento aggiunto.
Il codice necessario alla creazione di una chiave primaria è meno immediato ma non particolarmente complesso. La proprietà **PrimaryKey** dell'oggetto **DataTable** non richiede un **DataColumn**, bensì un array di **DataColumn.**
L'apparente complicazione è motivata dall'esigenza, talvolta, di creare delle chiavi composte basate su più colonne. Dobbiamo quindi instanziare un array di **DataColumn** costituito da 1 solo elemento, al quale assegnare il nostro **dcGuestbook.**
A questo punto l'array conterrà solamente il **DataColumn** che si riferisce alla colonna id e non dovremo far altro che assegnare alla proprietà **PrimaryKey** tale array:

```
Dim arrPK(1) As DataColumn
arrPK(0) = dcGuestbook
dtGuestbook.PrimaryKey = arrPK
```

Abbiamo definito la struttura del dataset in maniera tale da contenere le informazioni che ci servono. Il passo successivo consiste nell'inserire tale struttura in un file xml. Uno **Schema** è un insieme ben definito di regole che definiscono gli elementi e gli attributi di un file xml, al fine di definire con esatezza la validità e la consistenza dei dati. Lo schema può essere scritto in un file **.xsd** richiamato nel file xml, oppure essere parte interna dello stesso file xml e in tal caso è detto inline. Sfruttando il metodo **WriteXml** dell'oggetto **DataSet** potremo sfruttare quest'ultimo approccio per "trasferire" il dataset, la sua struttura e i suoi vincoli (constrains) nel file xml:

```
    dsGuestbook.WriteXml(strPath, XMLWriteMode.WriteSchema)
End Sub
```

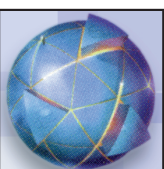
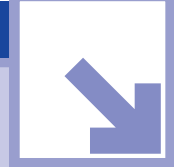
Inizialmente, quando il file **guestbook.xml** è stato appena creato, appare in questo modo:

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
```

```
  <xs:schema id="NewDataSet" xmlns="" xmlns:xs=
        "http://www.w3.org/2001/XMLSchema" xmlns:msdata =
                "urn:schemas-microsoft-com:xml-msdata">
   <xs:element name="NewDataSet" msdata:IsDataSet=
                "true" msdata:Locale="it-IT">
    <xs:complexType>
     <xs:choice maxOccurs="unbounded">
      <xs:element name="messaggio">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="id" msdata:AutoIncrement="true"
              msdata:AutoIncrementSeed="1" type="xs:int" />
         <xs:element name="crono" type="xs:dateTime"
                                   minOccurs="0" />
         <xs:element name="autore" type="xs:string"
                                   minOccurs="0" />
         <xs:element name="email" type="xs:string"
                                   minOccurs="0" />
         <xs:element name="oggetto" type="xs:string"
                                   minOccurs="0" />
         <xs:element name="commento" type="xs:string"
                                   minOccurs="0" />
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:choice>
    </xs:complexType>
    <xs:unique name="Constraint1" msdata:PrimaryKey="true">
     <xs:selector xpath=".//messaggio" />
     <xs:field xpath="id" />
    </xs:unique>
   </xs:element>
  </xs:schema>
</NewDataSet>
```

Salvo alcuni punti più complessi a prima vista, anche chi non conosce affatto xml, si renderà conto di come tale codice rispecchi perfettamente la struttura della tabella da noi definita. La terza routine della pagina è **btnInserisci_Click**, che gestisce l'evento di **click** sul pulante di invio commento:

```
Sub btnInserisci_Click(sender As Object, e As EventArgs)
  Try
    dsGuestbook = New DataSet()
    dsGuestbook.ReadXml(strPath, XmlReadMode.ReadSchema)
```

Inizialmente la procedura legge le informazioni contenute nel file xml, il suo schema, e lo memorizza in un'istanza di **DataSet** mediante il metodo **ReadXml.** Successivamente viene istanziato un **DataRow drGuestbook** per l'inserimento di una nuova entry nel **DataSet.**
Vengono effettuati degli assegnamenti per ogni colonna salvo per il campo id che viene "auto-generato".

I valori inseriti dall'utente nelle textbox sono assegnati ai rispettivi campi, mentre per la colonna "crono", ci si affida all'orario generato dal sistema mediante l'istruzione **DateTime.Now()**:

```
    Dim drGuestbook As DataRow=dsGuestbook.Tables(0).NewRow()
        drGuestbook("crono") = DateTime.Now()
        drGuestbook("autore") = txtAutore.Text.toString()
        drGuestbook("email") = txtEmail.Text.toString()
        drGuestbook("oggetto") = txtOggetto.Text.toString()
        drGuestbook("commento") = txtCommento.Text.toString()
        dsGuestbook.Tables(0).Rows.Add(drGuestbook)
```

Inserito nel dataset il nuovo commento, richiamiamo il solito metodo per la scrittura del file xml secondo i criteri di validità stabiliti nello schema:

```
    dsGuestbook.WriteXml(strPath, XMLWriteMode.WriteSchema)
```

La parte finale della procedura si occupa della visualizzazione di un messaggio di ringraziamento al posto della scritta "Firma il Guestbook" e di rendere invisibile il pannello pnlInizio contenete il form di inserimento. Tutto il codice della procedura è posto in un blocco Try-Catch al fine di intercettare e segnalare, con la solita label di conferma, anche eventuali errori.

```
        lblInizio.text = "Grazie " & txtAutore.Text.toString() & "!"
        pnlInizio.Visible = False
    Catch err As System.Exception
        lblInizio.text = err.message
    End Try

End Sub
```

L'ultima operazione da compiere è quella di controllare l'input inserito dall'utente mediante dei **RequiredFieldValidator** e **RegularExpressionValidator** che segnalano con un carattere di asterisco (*) gli input non validi. Le textbox **txtAutore** e **txtOggetto** sono dei campi richiesti, per cui dovremo inserire due controlli nel codice:

```
<asp:RequiredFieldValidator id="RequiredFieldValidator1"
        runat= "server" display="static" ControlToValidate=
                "txtAutore"> *</asp:RequiredFieldValidator>

<asp:RequiredFieldValidator id="RequiredFieldValidator2"
        runat= "server" display="static" ControlToValidate=
                "txtOggetto"> *</asp:RequiredFieldValidator>
```

Per quanto riguarda la **txtEmail**, dovremo controllare solamente che l'indirizzo inserito non sia formalmente errato, per cui facciamo seguire al codice della txtEmail, il seguente:

```
<asp:RegularExpressionValidator id=
            "RegularExpressionValidator1" runat="server"
ControlToValidate="txtEmail" Display="Static"
ValidationExpression="[\w-]+@([\w-]+\.)+[\w-]+">*
                    </asp:RegularExpressionValidator>
```

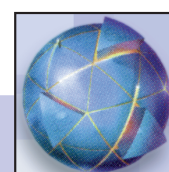
Il criptico valore della **ValidationExpression** si limita a controllare la presenza di un carattere di "@" e di un punto, in maniera ordinata ma non adiacente, all'interno della stringa inserita.
La mancanza di un **RequiredFieldValidator** indica la volontà di permettere all'utente di non inserire la propria e-mail, mentre il controllo appena esposto delimita inserimenti scorretti involontari. Esistono controlli più restrittivi sull'input di un campo e-mail, ma tali nozioni esulano dagli scopi dell'articolo.

Fig. 2 - Se ci sono degli errori nel completamento dei campi da parte dell'utente, i validator segnalano l'errore con un carattere asterisco rosso

La pagina **aggiungi.aspx** è così completata e funzionante. Ad ogni commento inserito correttamente, il file xml si arricchisce di un "record" che sarà simile nel formato, al seguente esempio:

```
<messaggio>
  <id>25</id>
  <crono>2002-11-27T20:30:13.9364304+01:00</crono>
  <autore>Marina Flamini</autore>
  <email>suaemail@prova.it</email>
  <oggetto>Prova di commento</oggetto>
  <commento>Continuate così! Dio vi benedica!</commento>
</messaggio>
```

LA PAGINA VISUALIZZA.ASPX

Completata la parte inerente all'inserimento dei commenti, non resta che realizzare la pagina di visualizzazione. Il codice necessario è relativamente semplice se ci si avvale del controllo web **Repeater** e dell'oggetto **DataView**. Inserito il **Repeater** all'interno della pagina web, e gli import iniziali, la parte di codice necessaria si riduce alle poche righe seguenti:

```
Sub Page_Load(sender As Object, e As EventArgs)
    strPath = Server.MapPath("guestbook.xml")
    If Not Page.IsPostBack Then
        If File.Exists(strPath) Then
            Visualizza()
        End If
    End If
End Sub

Sub Visualizza()
    dsGuestbook = New DataSet()
    dsGuestbook.ReadXml(strPath, XmlReadMode.ReadSchema)
    Dim dvGuestbook As DataView =
                    dsGuestbook.Tables(0).DefaultView
    dvGuestbook.Sort = "id DESC"
```

La procedura **Visualizza**, invocata al caricamento della pagina si occupa dell'associazione del **DataSet** al **Repeater**. Le prime due istruzioni di tale routine memorizzano nell'istanza **dsGuestbook**, il **DataSet** contenuto nel file **guestbook.xml**. Successivamente viene istanziato un oggetto **DataView** (**dvGuestbook**) mediante l'associazione con la tabella "messaggio" (di indice 0) del **DataSet**. L'assegnamento **dvGuestbook.Sort = "id DESC"** permette di visualizzare i messaggi a partire dall'ultimo inserito, secondo l'ordine cronologico (decrescente rispetto a id).
Alla proprietà DataSource del repeater (denominato **rptGuestbook**) viene assegnato il **DataView dvGuestbook** ed infine invocando il metodo **DataBind** del repeater avviene l'associazione dei dati:

```
    rptGuestbook.DataSource = dvGuestbook
    rptGuestbook.DataBind()
End Sub
```

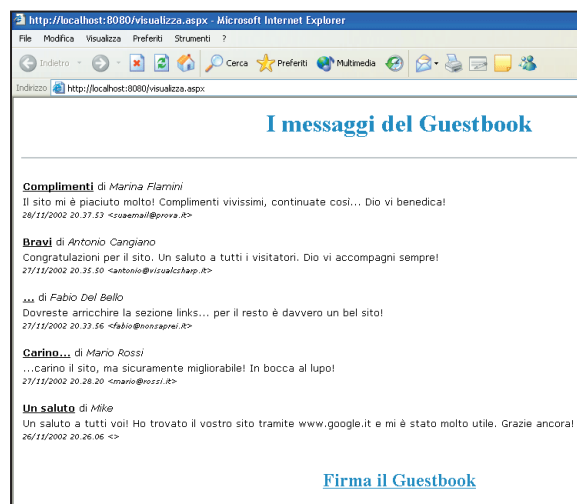
Il codice html, misto a codice ASP.NET, per la visualizzazione dei dati è il seguente:

```
<asp:Repeater id="rptGuestbook" runat="server">
  <ItemTemplate>
  <tr>
      <td width="100%">
     <u><b><%# DataBinder.Eval(Container.DataItem,
                    "oggetto") %></b></u> di <i>
         <%# DataBinder.Eval(Container.DataItem, "autore") %></i>
       </td>
       </tr>
       <tr>
         <td width="100%">
         <%# DataBinder.Eval(Container.DataItem, "commento") %>
       <br />
         <i> <font size= "1"><%# DataBinder.Eval(
                    Container.DataItem, "crono") %> &lt;
         <%# DataBinder.Eval(Container.DataItem, "email") %>&gt;
                    </font></i>
         <br />
         <br />
       </td>
     </tr>
   </ItemTemplate>
```

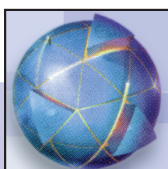
Tralasciando le possibili personalizzazioni del codice HTML, si ponga l'accento sulle istruzioni di associazione tra le informazioni presenti nei campi delle tabella del dataset e la pagina generata.
L'istruzione generica:

```
<%# DataBinder.Eval(Container.DataItem, "nomecampo") %>
```

inserisce il valore di nomecampo all'interno del codice html. Grazie all'oggetto **Repeater**, tutti i commenti presenti nel file xml, saranno visualizzati in maniera simile a quella mostrata in **Fig. 3**.

Fig. 3 - La pagina visualizza.aspx in presenza di alcuni commenti fittizzi

CONCLUSIONI

ADO.NET fornisce un ottimo supporto per l'uso di Xml come soluzione di data-storing veloce e diretta. La semplicità con la quale .NET permette di interagire con Xml, apre nuove prospettive per l'impiego alternativo, in quelle applicazioni in cui la mole di dati non è tale da richiedere database del calibro di Microsoft Sql Server.
Il guestbook presentato è volutamente semplice e personalizzabile, ma è davvero interessante notare come grazie a dotnet, un'applicazione web basata su XML non richieda la scrittura di una sola riga di codice in questo linguaggio.

Antonio Cangiano

FLASH e JSP

Realizzare una chat con i più avanzati strumenti disponibili.

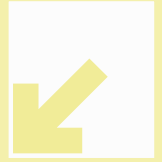
SUL CD
\soft\codice
\java_flash.zip

Se avete gia sviluppato applicazioni per internet, e visto che leggete questa rivista probabilmente sì, conoscerete sicuramente almeno un linguaggio di script lato server. Saprete quindi quali enormi vantaggi essi portino nella realizzazione di siti internet, consentendo di poter modificare dinamicamente i contenuti delle pagine al cambiare delle richieste effettuate. La realizzazione di una qualsiasi applicazione internet sarebbe di fatto impossibile senza il loro impiego. Ma, come spesso accade nell'informatica, ad evidenti vantaggi di una tecnologia, corrispondono carenze spesso legate proprio alle stesse caratteristiche positive. Nel caso dei linguaggi server side, il più grande scoglio che si incontra è l'impossibilità di effettuare operazioni che riguardino il lato client dell'applicazione, con la conseguenza di dover sempre riportare i dati sul server per poter effettuare anche la più semplice operazione di controllo. Per questo occorre spesso utilizzare uno strumento aggiuntivo, lato client, che permetta di rendere lo scambio di dati meno ridondante e che velocizzi quindi l'applicazione che si sta realizzando. In tutti e due i casi, lato client e lato server, la scelta dei prodotti per lo sviluppo è sufficientemente vasta da permettere a chiunque di trovare la soluzione ideale rispetto alle proprie conoscenze. In questo articolo tratteremo due delle possibili risoluzioni per lo sviluppo di contenuti web interattivi, riprendendo di fatto il discorso sulla programmazione in java server side, ed introducendo anche lo studio di interfacce utente realizzate in Flash.

IL PROGETTO

L'esempio che realizzeremo questo mese, pur essendo di semplice realizzazione, dimostra in maniera evidente l'efficienza dell'utilizzo di due linguaggi differenti per la nostra applicazione client/server. Prendendo infatti Java per la parte server del nostro software, abbiamo la possibilità di usufruire di un linguaggio potente ed immediato come JSP che facilita di molto la creazione di ciò che andremo a realizzare, mentre, lato client, l'adozione di uno strumento come Flash, rende la nostra applicazione ricca di possibilità importanti per un ambiente client, come ad esempio, nel nostro caso, la possibilità di caricare dei dati senza dover rieffettuare il refresh della pagina. Ciò che andremo a realizzare in questo articolo, una chat, probabilmente sarà gia stata vista da molti di voi come esempio in altre pubblicazioni per imparare ad utilizzare il proprio linguaggio di script server side, ma, ciò che probabilmente non è stato ancora visto, è come una applicazione di questo tipo, tra l'altro perfettamente funzionante anche solo lato server, possa trarre beneficio dall'utilizzo di tecnologie differenti per ciascuno dei due lati della nostra applicazione. Flash infatti mette a disposizione lato client due importanti possibilità:

- **Refresh automatico** - Essendo il filmato flash eseguito da un plug-in interno al browser, esso è in effetti svincolato dalla pagina html statica, e rimane in esecuzione anche dopo che la pagina stessa è stata completamente caricata. Questo ci permette di avere un ambiente di sviluppo che, oltre ad intercettare eventuali azioni dell'utente, può compiere esso stesso delle operazioni determinate dallo scorrere del tempo, come ad esempio effettuare il refresh di una chat per visualizzarne i contenuti aggiornati.

- **Caricamento trasparente** – Flash possiede una particolare caratteristica che gli permette di caricare dei dati da file di testo in maniera completamente invisibile all'utente. Questo significa che, al verificarsi di determinate condizioni, è possibile tramite Flash aggiornare il contenuto di alcune variabili interne per aggiornare il filmato senza che quest'ultimo debba essere ricaricato. Il risultato è, nel nostro caso, una chat contenuta in una pagina che non viene mai aggiornata, risultando di fatto in un'interfaccia molto veloce e graficamente migliore.

Con queste potenti caratteristiche, la nostra chat si arricchirà di contenuti sicuramente nuovi rispetto agli esempi che potete aver già incontrato, e vi porterà a scoprire i vantaggi dello scegliere le tecnologie adatte al tipo di applicativo che si intende realizzare.
Ma cominciamo finalmente a digitare codice.

SERVER JSP

Come premesso, il server della nostra applicazione verrà realizzato utilizzando forse il più semplice strumento per la realizzazione di contenuti Java per internet: JSP. Nel numero scorso abbiamo dato un'introduzione al linguaggio, notando che la sintassi non è dissimile dal Java che noi tutti siamo abituati ad utilizzare e che, in fondo, esso non è che uno strato che nasconde una tecnologia Java pura al 100% che prende il nome di **Servlet**. Non essendo questo un corso che porta all'apprendimento di un particolare linguaggio, quanto piuttosto una fase di introduzione e sperimentazione dei linguaggi conosciuti, non ci soffermeremo molto sulla sintassi del codice di esempio, ponendo invece l'attenzione sul risultato del nostro lavoro e sulle sue caratteristiche. Iniziamo comunque creando un file con estensione JSP nella directory di root del nostro server web e digitandovi il codice presente sul CD alle-

gato nel file: **Mia Chat.jsp.** Andiamo ad esaminare brevemente il listato sopra riportato analizzando il risultato restituito in output. La prima istruzione, nel primo blocco, crea un'array di stringhe che utilizzeremo come buffer per la lista di messaggi che verranno via via immessi. La funzione che segue, molto particolare in JSP, viene richiamata ogni prima volta che una servlet viene avviata, e serve di fatto ad inizializzare l'array appena creato immettendovi delle stringhe vuote. Una volta realizzata la struttura dati necessaria alla nostra applicazione, il passo successivo consiste nell'elaborare le richieste di dati che verranno di volta in volta effettuate, ed il secondo blocco di codice esegue queste poche, semplici operazioni. Innanzitutto controlla se insieme alla richiesta di visualizzazione della pagina sono state inviate anche le variabili che corrispondono alla persona (nick) ed al messaggio (messaggio) di un'eventuale nuovo messaggio da aggiungere alla lista. Se ciò è stato fatto, allora aggiunge il nuovo messaggio alla lista formattandolo in una stringa in formato html per poi proseguire oltre. Le istruzioni successive si occupano dell'output dei dati contenuti nel nostro buffer, ed altro non sono che delle istruzioni di stampa delle stringhe già precedentemente formattate in html. Ma prima di ciò, affinché il risultato della stampa possa essere analizzato dal filmato Flash, il tutto và assegnato ad una variabile chiamata **messaggi** che utilizzeremo per recuperare i dati ricevuti all'interno del filmato client. Il testo deve essere infatti formattato, per poter essere caricato correttamente dal filmato Flash, secondo le specifiche del formato **application/x-www-form-urlencoded** che, a dispetto della sigla piuttosto poco amichevole, altri non è che lo standard per il passaggio di parametri in internet tramite le stringhe di interrogazione visibili nella barra di indirizzi del browser (il metodo **GET** dell'Http). In breve, è sufficiente indicare il nome della variabile della quale si intende definire il valore preceduta da una e commerciale (&), seguita da un segno di uguale (=) e dal valore che si intende dare alla variabile stessa, ad esempio, la stringa seguente carica due variabili di nome regione e stato, ed imposta per ciascuna i valori "**lazio**" ed "**italia**":

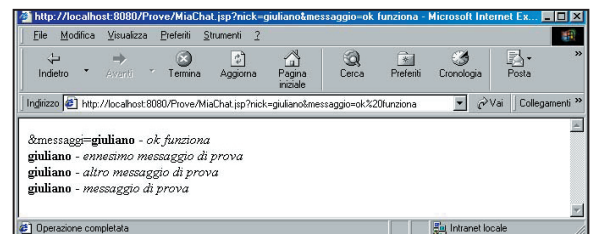
```
&regione=lazio&stato=italia
```

Questo tipo di stringa può essere tranquillamente utilizzata per passare, ad un filmato Flash che le richieda, le due variabili **regione** e **stato**. Ma testiamo il nostro file JSP. Già, perché il semplice listato appena scritto è effettivamente un motore per chat perfettamente funzionante del quale possiamo verificare l'effettiva bontà semplicemente collegandosi al proprio server web e digitando l'indirizzo del file JSP creato. Ma ad un primo caricamento della pagina in questione non appare null'altro che la stringa **&messaggi=**. Questo dipende dal fatto che il buffer utilizzato per contenere i messaggi inviati alla chat, essendo la prima volta che essa viene caricata, è completamente vuoto, e quindi è impossibile visualizzare alcunché. Per fare in modo che l'array di stringhe che contiene i messaggi della chat si riempia con i primi dati, occorre fare in modo che la pagina li riceva e li aggiunga quindi alla lista. Per ottenere questo risultato, si devono passare le due variabili necessarie alla composizione del messaggio (**nick** e **messaggio**) attraverso la stringa di interrogazione, così da fare in modo da renderle disponibili alla pagina JSP. Si provi ad inserire al termine dell'indirizzo del proprio file JSP la seguente stringa:

```
?nick=Giuliano&messaggio=messaggio+di+prova
```

Ora l'indirizzo della pagina dovrebbe somigliare a questo:

```
http://localhost:8080/NomeCartella/NomeFile.jsp?nick=
giuliano&messaggio=messaggio+di+prova
```

Fig. 1 - Aggiunta di un nuovo messaggio.

Una volta lanciato di nuovo il browser al caricamento della pagina in questione, potete finalmente osservare che la nostra chat si arricchisce del primo messaggio lasciato dall'utente **Giuliano** contenente la stringa **"messaggio di prova"**. Ovviamente potete modificare tranquillamente il valore delle variabili per vedere come ogni volta viene inserito un messaggio differente, mantenendo comunque in memoria i messaggi precedenti. Potete inoltre avviare il vostro browser in una nuova finestra che punti alla stessa pagina JSP, e simulare due utenti che chattano effettivamente tra di loro, osservando che, all'immissione di un nuovo messaggio da parte di uno dei due, l'altro riesce a vederlo semplicemente aggiornando la pagina, come in **Fig.1.** Un'ultima osservazione importante consiste nel visualizzare il vero contenuto che il file JSP rilascia al browser osservando il codice html della pagina. Per farlo, se utilizzate Internet Explorer cliccate sul menù **Visualizza** e scegliete la voce **Html**. Come potete osservare, il file, visto in questo modo, appare molto differente dalla visualizzazione nel browser per la presenza dei numerosi tag Html che indicano come formattare le informazioni contenutevi. Sebbene fosse possibile eliminare questi tag, utili esclusivamente per la visualizzazione in un browser web, ho scelto di mantenerli per illustrarvi, successivamente, una importante caratteristica di Flash rinnovata nella sua versione MX, e cioè la capacità non solo di caricare variabili di testo da un file qualsiasi, ma anche la possibilità di interpretarne eventuali formattazioni Html. Bene, detto ciò, e testato il corretto funzionamento del nostro motore di chat, occorre passare alla fase più artistica del progetto, che consiste nel dotare di un'interfaccia grafica

l'applicazione da noi sviluppata. Per questo si rende necessario abbandonare JSP per tuffarci nella programmazione di un client in Flash. Avviate quindi il programma e proseguiamo.

CLIENT FLASH

Il lato Flash della nostra applicazione, per chi conosce il prodotto, è veramente banale da realizzare. E' infatti sufficiente realizzare graficamente una form che renda possibile l'immissione di nuovi messaggi e la visualizzazione di quelli contenuti nel buffer del server, oltre, successivamente, ad indicare quando caricare i messaggi aggiornati ed il file da cui prelevare i messaggi stessi. Ma, siccome credo che la spiegazione delle azioni che il client deve effettuare sia più difficile dell'effettiva realizzazione delle stesse, passiamo senza indugiare oltre alla creazione dell' altro lato della nostra applicazione. Il primo passo consiste nel realizzare, come detto precedentemente, una form per contenere i dati che ci sarà utile visualizzare. A questo scopo si devono inserire, sullo stage del filmato, diverse aree di testo, mantenendo un'impostazione simile a quella della **Figura 2**. Come è possibile osservare dall'immagine, vi sono 3 campi di testo di differente grandezza tutti piazzati all'interno dello stage. I nomi di ognuno dei campi di testo, da inserire nel pannello **Proprietà** dove appare la stringa **<Instance Name>**, sono rispettivamente:

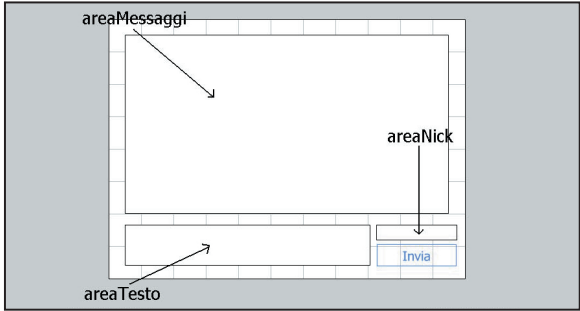



Fig. 2 - Impostazioni dello Stage

- **areaMessaggi** – La casella di testo più grande, che conterrà tutta la lista di messaggi immessi dai vari utenti della chat e che deve essere in grado di interpretare testo formattato in Html.
- **areaTesto** – La casella di testo in basso a sinistra che deve contenere il testo del nuovo messaggio da inviare alla chat.
- **areaNick** – La casella più piccola in basso a destra che deve invece contenere il nome dell'utente che sta mandando il messaggio.

Oltre ai campi di testo, è presente anche un pulsante che permette di inviare un nuovo messaggio composto. Una volta realizzata questa semplice struttura è conveniente farne una MovieClip che possa rispondere a dei particolari eventi che ci è utile intercettare, ed a tale scopo, selezionate tutto il contenuto dello stage e premete il tasto **F8.** Una volta completato questo passo, selezionate la MovieClip appena realizzata ed assegnatele il nome "**chat**" dal pannello **Proprietà.** Bene, fatto ciò il contenuto grafico della nostra chat è completato, quello che ci rimane da fare ora è implementare da codice delle istruzioni che ci permettano di collegare il nostro client con il server realizzando nel contempo un sistema di refresh automatico. La prima parte di codice va inserita nella MovieClip "**chat**", che sarebbe poi quella che contiene tutti i campi di testo utili alla nostra applicazione. Per procedere all'immissione è necessario selezionare la **Movie** cliccandoci sopra e successivamente aprire il pannello **Azioni** impostando la modalità **Esperto.** Una volta compiuti questi passi occorre digitare il seguente codice:

```
onClipEvent (load) {
  attesa=20;
  trascorso=0;
  areaTesto.text="testo";
  areaNick.text="nick";
  areaMessaggi.text="Inizializzazione in corso...";
  this.loadVariables("MiaChat.jsp"); }
onClipEvent (data) {
  areaMessaggi.html=true;
  areaMessaggi.htmlText = messaggi; }
onClipEvent (enterFrame) {
  if((trascorso++)==attesa){
    trascorso=0;
    this.loadVariables("MiaChat.jsp");  } }
```

Il listato riportato si occupa di dichiarare tre funzioni che svolgono dei compiti molto particolari in Flash. La prima viene richiamata quando il filmato viene caricato per la prima volta, ed in questo caso viene utilizzata per dichiarare ed inizializzare alcune variabili, per riempire i campi di testo con dei valori di default nonché caricare il contenuto del buffer della chat in memoria. La seconda funzione invece viene utilizzata da Flash ogni qual volta dei nuovi dati vengono caricati in memoria, e nel nostro caso serve ad immettere il contenuto restituito dal file JSP nel campo di testo di nome **areaMessaggi**, specificando, nel contempo, che il testo dichiarato deve essere analizzato in formato Html. La variabile messaggi utilizzata nella seconda istruzione di questa funzione, altri non è che quella dichiarata dal file JSP alla quale abbiamo assegnato il contenuto di tutti i messaggi del buffer della chat. La terza ed ultima funzione viene richiamata ogni volta che la MovieClip entra in uno dei suoi frame. Questo significa che se essa è in loop, e lo è per default , ogni tot di tempo essa cambierà frame e questa funzione verrà richiamata, dandoci così l'opportunità di inserire al suo interno tutti quei comportamenti che debbono essere eseguiti in maniera ciclica. In questa funzione inseriamo infatti il refresh della pagina semplicemente immettendo un controllo sul tempo di attesa tra un

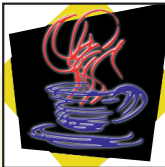
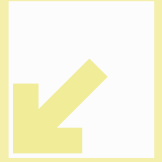
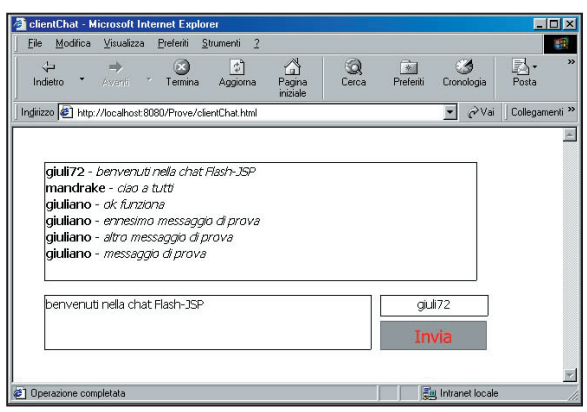



Fig. 3 - Il Benvenuto nella chat.

refresh e l'altro (con le variabili **trascorso** ed **attesa**), e l'istruzione per il caricamento della nuova variabile **messaggi** con il contenuto della chat aggiornato. Quando i dati saranno stati caricati, la funzione **onClipEvent(data)** si preoccuperà di aggiornare la casella di testo che visualizzerà gli ultimi dati acquisiti. Il passaggio successivo consiste nell'immettere dell'altro codice, associato al pulsante **invia**, che permetta di trasmettere dei nuovi messaggi alla pagina JSP, in modo che essa possa aggiungerli alla lista. Occorre ricordare infatti che, caricando la pagina del server JSP senza inviare alcuna variabile, cioè come abbiamo fatto sin ora, il codice java altro non fa che rilasciare il contenuto dell'array di stringhe in memoria, senza modificarlo. Per aggiungere messaggi si devono invece fornire alla pagina due valori, **nick** e **messaggio**, che essa possa utilizzare per formattare un nuovo messaggio. Andiamo quindi ad inserire il codice necessario. Fate doppio click sulla **MovieClip chat** in modo da posizionarvi al suo interno, dopo di che selezionate il pulsante invia cliccandovi sopra. Aprite quindi di nuovo il pannello **Azioni** per andare a scrivere il listato seguente:

CODICE

Il codice mostrato, dichiara una funzione che viene richiamata ogni volta che il pulsante viene premuto. Essa copia il contenuto dei due campi di testo dove si è immesso il nick ed il messaggio in due varabili della MovieClip, che vengono poi inviate dal metodo loadVariables, utilizzato stavolta non solo per il caricamento della variabile messaggi

```
on (press) {
  this.nick = areaNick.text;
  this.messaggio = areaTesto.text;
  loadVariables("MiaChat.jsp",this,"GET"); }
```

Il codice mostrato, dichiara una funzione che viene richiamata ogni volta che il pulsante viene premuto. Essa copia il contenuto dei due campi di testo dove si è immesso il nick ed il messaggio in due varabili della MovieClip, che vengono poi inviate dal metodo **loadVariables**, utilizzato stavolta non solo per il caricamento della variabile messaggi. Le due variabili quindi, rese note alla pagina JSP, entrano a far parte della lista dei messaggi e possono essere correttamente visualizzate. Sfruttando poi la capacità di caricamento automatico dei dati vista in precedenza, il client và ad aggiornare, dopo un lasso di tempo predefinito, la casella che deve contenere i messaggi, visualizzando il nuovo messaggio spedito in maniera del tutto invisibile all'utente. Bene, ora che il nostro programma è terminato non ci resta che testare il suo corretto funzionamento. A questo scopo, salvate il file creato con Flash nella stessa cartella del file JSP e pubblicate il filmato che utilizzeremo poi come client. Nella pubblicazione è importante prevedere anche la generazione di un file Html che funzioni da contenitore per il filmato, ad evitare che quest'ultimo venga eseguito nel player Macromedia. Una volta in possesso del file .html tutto ciò che occorre fare è caricarlo nel browser e iniziare ad utilizzarlo. Ovviamente il caricamento non potrà avvenire in maniera diretta (con un doppio click sul file in questione) poiché in quel caso esso verrebbe trattato come un file presente sul sistema, mentre a noi interessa che il nostro server web a processi il file JSP che deve fornirci i dati relativi ai messaggi. Per questo motivo occorre caricare il file digitando nella barra degli indirizzi del browser una stringa simile alla seguente:

```
http://localhost:8080/NomeCartella/NomeFile.html
```

Questo indirizzo ci connette alla pagina del client Flash di elaborando il file JSP realizzato. Altra operazione importante da compiere è evitare che la pagina venga memorizzata nella cache del browser. Questo risulterebbe in un errore di difficile individuazione che non ci permetterebbe di sincronizzare i messaggi inviati con la loro corretta visualizzazione. Per evitare ciò, impostate nel vostro browser il nuovo caricamento della pagina ogni qual volta essa deve essere visualizzata, in Internet Explorer ad esempio, dal menù **Strumenti->Opzioni Internet** selezionate la scheda **Generale** e premete il tasto **Elimina File** rispondendo **Ok** alla dialog che vi verrà proposta. Successivamente premete il pulsante **Impostazioni** e selezionate nel gruppo **"Ricerca versioni più recenti delle pagine memorizzate"** l'opzione **"All'apertura della pagina"**, premendo successivamente il tasto **Ok.** Compiuta quest'ultima operazione, potete verificare finalmente che la chat realizzata assolve gli scopi che ci eravamo prefissati, andando a svolgere esattamente il compito che le avevamo assegnato.

CONCLUSIONI

In questo articolo abbiamo potuto osservare che utilizzando due strumenti diversi per il lato client e server di una applicazione web, si possono sfruttare le caratteristiche particolari di più prodotti per soddisfare le proprie esigenze nel migliore dei modi. Questo porta, nella maggioranza dei casi, ad una maggiore efficienza del prodotto ed ad un minore tempo di sviluppo. La morale è quindi: non cercate di sviluppare applicazioni nell'unico modo in cui siete capaci, piuttosto analizzate tutti gli strumenti possibili, scegliete quello più adatto al vostro problema e non abbiate paura mai di cimentarvi nell'apprendimento di qualcosa di nuovo. Alla prossima.

Giuliano Uboldi

Creiamo la sezione protetta del nostro sito

Quante volte, sfogliando pagine internet, vi è capitato di trovarne alcune che, per essere visualizzate, richiedevano autenticazione con nome utente e password?

SUL CD
\soft\codice\
sezioneprotetta_source.zip

La protezione di pagine web dall'accesso incondizionato degli utenti è ormai divenuta caratteristica comune a molti siti internet, a partire dal sito amatoriale fino ad arrivare al portale di e-businnes o di informazione. L'obiettivo di chi attua una limitazione del genere non è univoco. Per esempio, si può voler proteggere un'area del proprio sito web per celare informazioni riservate o elementi di amministrazione del sito stesso; oppure si può avere l'esigenza di creare un'area riservata ai soli clienti che abbiano acquistato un particolare servizio, fruibile proprio dalle pagine ad accesso limitato.

LA STRUTTURA

La struttura dell'applicazione che andiamo a realizzare è molto basilare (in modo da permettere una migliore comprensione di tutti i passaggi della sua creazione) ma, al contempo, sarà completamente funzionale ed utilizzabile così come è presentata.
Al termine di questa lettura anche voi sarete in grado di implementare un'applicazione completa per la gestione di un'area ad accesso limitato al vostro sito web. L'applicazione verrà suddivisa in differenti file .asp ognuno destinato ad una particolare funzionalità. Schematicamente, la nostra area protetta è costituita da una pagina, **loginform.asp**, che visualizza il form per l'inserimento di email e password, form che viene inviato a **login.asp** che rappresenta il cuore dell'applicazione; tale pagina processa i dati inviati dall'utente e, se corretti, esegue due operazioni molto importanti:

1) istanzia una variabile di sessione, una specie di flag che indica che l'utente si è autenticato correttamente;

2) redireziona l'utente alla pagina **riservato.asp**.

riservato.asp è la pagina a contenuto effettivamente riservato del nostro sito; può essere una pagina singola (come nell'esempio presentato in questo articolo) o anche una serie di pagine web; in ogni caso, il denominatore comune di queste pagine riservate è il controllo sull'esistenza o meno della variabile di sessione di cui sopra. Se la variabile non è stata istanziata, significa che l'utente sta operando un accesso non autorizzato, tentando di scavalcare il form di autenticazione e l'autenticazione stessa. Ora analizziamo nel dettagli tutti gli elementi che costituiscono l'applicazione e cioè il database **sezioneprotetta.mdb**, destinato a contenere i dati relativi ai nostri utenti, e le tre pagine asp le cui funzionalità sono state solo sommariamente presentate.

Fig. 1 - Creazione del database sezioneprotetta.mdb e della tabella users.

IL DATABASE

Per prima cosa è necessario creare il database in cui memorizzare email e password di tutti coloro a cui è consentito l'accesso nell'area riservata. Creiamo un database chiamato **sezioneprotetta.mdb**, apriamolo e creiamo (tramite la visualizzazione struttura) una tabella, che chiameremo **users**, costituita da tre campi:

- **ID:** di tipo intero, contatore; sarà la chiave primaria della tabella.
- **email:** conterrà l'indirizzo email di ogni utente.
- **password:** conterrà la password relativa ad ogni utente

In **Fig. 1** è mostrato ciò che dovreste ottenere durante la creazione del database tramite MS Access 97/2000: ora popoliamo la tabella inserendo alcuni indirizzi email e alcune password casuali, il database di supporto potete trovarlo già pronto fra il software relativo a questo articolo incluso nel CD-ROM.

LOGINFORM.ASP

E’ il punto da cui i nostri utenti potranno iniziare la procedura d’accesso all’area riservata del sito.

La pagina visualizza un form in cui sono presenti due campi **<input>** per l’inserimento di email e password e un bottone che, una volta premuto, invia il contenuto del form a **login.asp**.

Il codice (quasi tutto HTML) di **loginform.asp** è molto semplice:

> **APICI**
>
> **Nell’utilizzo di stringhe inserite dall’utente in comandi SQL è importante sostituire sempre gli apici singoli con un doppio apice singolo:**
>
> Replace(varStringa, “’”, “’’”)

```
<html>
<head>
  <title> LoginForm - Sezione Protetta </title>
</head>
<body>
<table border=0 bgcolor=#006060 cellspacing=1 width=150
          cellpadding=0 style="font-size:10px; color:#000000;
                                        font-family:Verdana">
  <tr>
    <td align=center bgcolor=#B4C4D3 height=20><b>LOGIN
                                              FORM</b></td>
  </tr>
  <tr>
    <td bgcolor=#E2E8E9>
    <table border=0 cellspacing=0 cellpadding=0 width=100%
      style="font-size:10px; color:#000000; font-family:Verdana">
      <form method=post action=login.asp>
      <tr>
        <td> EMAIL:<br> <input type=text name=
                   email class=input3D size=18 value=""></td>
      </tr>
      <tr>
        <td> PASSWORD:<br> <input type=password
                   name=password size=18 value=""></td>
      </tr>
      <tr>
        <td align=center><input type=submit value="LOGIN >>">
                                                        </td>
      </tr>
      <tr><td height=6></td></tr>
<%
'messaggio d'errore da visualizzare?
if len(request.querystring("err"))>0 then
%>
      <tr>
        <td align=center>
        <font color=#ff0000>EMAIL O PASSWORD ERRATE!</font>
        </td>
      </tr>
      <tr><td height=6></td></tr>
<%
end if
%>
    </table>
    </td>
  </tr>
  </form>
</table>
</body>
</html>
```

Come potete notare, dopo la visualizzazione dei campi testo e del bottone, viene effettuato un controllo sull’eventuale presenza dell’elemento **err** nella querystring. Come vedremo in seguito, nella pagina **login.asp**, se il controllo sui dati inseriti dall’utente non ha esito positivo, il browser viene redirezionato al form di login tramite il seguente **redirect:**

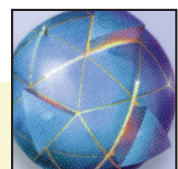
```
response.redirect("loginform.asp?err=true")
```

Nel caso dunque che nella querystring sia presente l’elemento **err**, in fondo al form viene visualizzato un messaggio d’errore che indica all’utente che l’indirizzo email e/o la password non sono corretti.

LOGIN.ASP

E’ il cuore dell’applicazione; qui “viene deciso” se l’utente può continuare la sua navigazione all’interno delle pagine protette o se, invece, non è autorizzato a procedere e viene quindi redirezionato al form di autenticazione iniziale.

Il codice di **login.asp** (in questo caso solo ASP) non è molto complesso, ma va analizzato con molta attenzione:

```
<%
'rileva i dati inviati dal form
dim email, password
email = replace(request.form("email"), "'", "''")
password = replace(request.form("password"), "'", "''")
dim cn, rs, sql
'connessione al database
set cn = Server.CreateObject("ADODB.Connection")
cn.Open "driver={Microsoft Access Driver (*.mdb)};dbq="
                    & Server.MapPath("sezioneprotetta.mdb")
sql = "SELECT ID FROM users WHERE EMAIL='"&email&"' AND
                                    PASSWORD='"&password&"'"
'cerca l'utente nel database
set rs = cn.execute(sql)
'utente trovato
if not rs.eof then
  session("userlogged") = true
  rs.close
  set rs = nothing
  cn.close
  set cn = nothing
  response.redirect "riservato.asp"
```

```
'utente non trovato
else
  rs.close
  set rs = nothing
  cn.close
  set cn = nothing
  response.redirect("loginform.asp?err=true")
end if
%>
```

Lo script **login.asp** raccoglie email e password provenienti dal form di **loginform.asp** e li memorizza nelle variabili **email** e **password.** Da notare che gli eventuali apici (') immessi dall'utente vengono sostituiti con un doppio apice (attenzione: due volte l'apice singolo, non le virgolette doppie!). Successivamente viene composta la stringa SQL che interroga il database cercando un record che abbia email e password corrispondenti a quelli immessi nel form. Se il recordset risultante è vuoto, significa che nel database non è stata trovata la corrispondenza delle due variabili e quindi l'utente non è autenticato; in caso contrario, l'utente è presente nel database, ed ha inserito la password correttamente. Nel primo caso, il navigatore viene reindirizzato alla pagina **loginform.asp;** nel secondo caso, invece, l'utente viene inviato alla pagina **riservato.asp** ma, prima di ciò, viene creata una variabile di sessione di tipo booleano denominata **"userlogged",** a cui viene assegnato il valore **true.** Una variabile di sessione è una variabile che, al contrario di quelle convenzionali dichiarate con la clausola **Dim** che hanno scope(visibilità) locale alla singola **pagina .asp**, hanno uno scope globale a livello della stessa sessione e il loro valore può essere quindi passato fra pagine **.asp** differenti. La creazione di questa variabile è fondamentale, in quanto, ad ogni accesso ad una pagina dell'area riservata (durante la medesima sessione), ne verrà verificata la presenza; nel caso essa non sia presente o non abbia valore **true**, il browser client verrà reindirizzato alla pagina contenente il form per il login. Tutto ciò può sembrare una futile precauzione, ma supponiamo che un navigatore "malintenzionato" conosca in qualche modo che la pagina riservata del nostro sito ha come url: **http://www.miosito.com/riservato .asp** a questo punto, per scavalcare la protezione, gli basterebbe digitare direttamente tale url senza passare attraverso la pagina di autenticazione del sito cosicchè la nostra "barriera" risulterebbe perfettamente inutile.
Controllando invece ad ogni pagina dell'area riservata la presenza della variabile di sessione **userlogged**, la protezione non potrà essere oltrepassata in maniera irregolare.

RISERVATO.ASP

Questa è l'unica pagina effettivamente protetta, pagina che, per essere accessibile, richiede che l'utente abbia eseguito e completato correttamente il passaggio di autenticazione. Per semplicità, in questa applicazione, la pagina riservata è soltanto una, ma possiamo immaginare di avere un numero molto superiore di pagine protette; il punto fondamentale è che, in testa ad ognuna di queste pagine, deve essere presente la porzione di codice che controlla l'avvenuta autenticazione dell'utente.
Vediamo in dettaglio il listato di **riservato.asp:**

```
<%
'se l'utente non si era autenticato redirezionalo al form di login
if not session("userlogged") then
  response.redirect("loginform.asp")
end if
'se l'utente precedentemente autenticato vuole uscire
if request.querystring("action")="logout" then
  session.abandon()
  response.redirect("loginform.asp")
end if
%>
<html>
<head>
  <title> Pagina principale - Sezione Protetta</title>
</head>
<body>
Utente autenticato.<br>
Stai visualizzando una pagina ad accesso riservato.<br><br>
<a href=main.asp?action=logout>Clicca qui per uscire.</a>
</body>
</html>
```

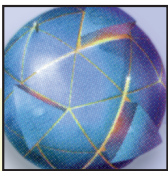
Nella parte iniziale viene eseguito il controllo sulla presenza della variabile di sessione. Se essa non è stata istanziata o se esiste ma non ha valore **true**, il client viene riportato alla pagina di visualizzazione del form di login senza che venga visualizzato alcunché delle informazioni riservate.
Seguono poi alcune righe di codice che permettono all'utente che si sia già autenticato di uscire dall'area riservata con relativa eliminazione di tutte le variabili di sessione (quindi anche della variabile **userlogged**) e redirezionamento al form di autenticazione. Questo codice viene eseguito, grazie al passaggio dell'elemento **"action"** (a cui viene assegnata la stringa **"logout"**) tramite la querystring, solo se l'utente clicca sul link testuale **("Clicca qui per uscire")** che comparirà in fondo alla pagina riservata. Il resto della pagina è codice HTML che, semplicemente, informa l'utente del fatto che l'autenticazione è avvenuta con successo e che si sta visualizzando una pagina ad accesso riservato. Come accennato, segue il link per "uscire" dall'area riservata.
In un prossimo articolo approfondiremo l'argomento con altri interessanti aspetti.

Paolo Capitani

VARIABILI DI SESSIONE

Una variabile di sessione è una variabile che ha scope relativo all'intera sessione, per cui il suo valore può essere letto (e/o impostato) anche da pagine .asp differenti, purchè nell'ambito della medesima sessione di navigazione.

ELEMENTI DELLA APPLICAZIONE

- **sezioneprotetta.mdb**
- **loginform.asp**
- **login.asp**
- **riservato.asp**

PHP & SMTP per spedire email dal web

La sempre maggiore diffusione di Internet e del WWW ha spinto ad integrare nel web anche uno dei servizi più "vecchi" ed utilizzati in rete: la posta elettronica…

SUL CD
\soft\codice\
php_smtp.zip

La configurazione di un programma di posta elettronica, come Outlook Express o Eudora, è una delle classiche operazioni che l'utente Internet svolge al momento della registrazione ad un ISP (Internet Service Provider) o ad uno dei numerosi siti che offrono mailbox gratuite. Il nuovo account di posta sarà costituito da: un indirizzo email, l'indirizzo del server per le posta in uscita (SMTP), di quello per la posta in arrivo (POP3) e l'username e la password per lettura della nostra mailbox. I problemi iniziano a sorgere quando non si ha la possibilità di utilizzare il proprio PC per inviare o ricevere email. Prima dello sviluppo di applicazioni mail per il web, l'utente doveva riconfigurare il proprio account di posta sul nuovo computer; cosa non del tutto immediata se, oltre ad avere di fronte un programma di posta a lui sconosciuto, diverso dal suo era anche l'ISP a cui il PC era collegato. In questo caso bisognava ricercare il nuovo indirizzo del SMTP visto che, per motivi di sicurezza, l'utilizzo del server è concesso ai soli clienti dell'Internet Service Provider. In definitiva, troppe erano le cose da fare e da ricordare per spedire una email e/o per effettuare un controllo veloce della propria casella di posta; che magari, dopotutto, risultava essere anche vuota. Oggi grazie allo sviluppo di applicazioni web dinamiche che si interfacciano ai server per la gestione della posta (SMTP e POP3), l'utente può inviare e ricevere email utilizzando un semplice browser web senza così effettuare alcun tipo di configurazione; gli unici dati che dovrà ricordare sono l'username e la password. In questo articolo, aiutandoci con la programmazione ad oggetti (inserita dalla ver. 4 del PHP) realizzeremo una classe chiamata smtp, che implementerà tutte le proprietà ed i metodi necessari per inviare email ad un server SMTP. Da premettere che in PHP esiste già una funzione di spedizione mail (**mail()**) che non gestisce però gli attach, lacuna che la nostra classe provvederà a colmare.

I SOCKET

Un socket non è altro che un modo di comunicare con altri programmi utilizzando descrittori di file standard di Unix. Un descrittore di file è semplicemente un intero associato ad un file aperto. In Unix tutto viene rappresentato attraverso i file: un FIFO, un terminale, un vero file su disco, una connessione di rete, etc. La funzione PHP *fsockopen* restituisce un descrittore di file associato ad una connessione TCP stabilita verso uno spefico port di un host Internet. Di conseguenza le operazioni di invio delle informazioni dal e verso il server corrisponderanno alla lettura (*fgets*) ed alla scrittura (*fputs*) del file.

FORMATO DEI MESSAGGI

Nei sistemi di posta elettronica, il messaggio è suddiviso in due parti: l'**intestazione** e il **corpo**. Il corpo è per il destinatario umano, mentre l'intestazione contiene informazioni di controllo per gli agenti utenti (client SMTP). È grazie ai campi di intestazione che i clienti di posta, come Outlook Express ed Eudora, riescono ad avere informazioni sul mittente, sui destinatari, sulla data di invio, sull'oggetto, sulla priorità e soprattutto sulla natura del corpo del messaggio. Ai tempi di ARPANET la posta elettronica consisteva esclusivamente di messaggi testuali espressi in ASCII. Questo tipo di approccio (dettato dall'RFC 822) diventò presto inadeguato, tramite email infatti, deve essere possibile: inviare e ricevere messaggi in lingue accentate, in alfabeti non latini (come ebraico e russo), in lingue senza alfabeto (come cinese e giapponese) e messaggi che non contengano assolutamente testo scritto (come audio e video). Una soluzione fu proposta con l'RFC 1341 (successivamente aggiornata con l'RFC 1521) e chiamata **MIME** (**Multipurpose Internet Mail Extension**). L'idea base di MIME è di continuare ad usare il formato RFC 822, ma di aggiungere una struttura al corpo del messaggio e di definire regole di codifica per i messaggi non ASCII. Così facendo, è possibile inviare i messaggi MIME usando i protocolli di posta elettronica esistenti, l'unica cosa che si è dovuto modificare sono stati i programmi per l'invio e la ricezione. Vediamo un esempio di come è strutturata una semplice mime mail:

```
--------------------------------------------------------------
busta
--------------------------------------------------------------
from: "antonio" joker@myoffice.it
to: emma@myoffice.it, adriano@interfree.it
Cc: radar@myoffice.it, spiderac@myoffice.it,
dodgio@myoffice.it
Ccn: iti@iti-cesena.it
--------------------------------------------------------------
intestazione del messaggio
--------------------------------------------------------------
from: joker@myoffice.itto: emma@myoffice.it, adriano@interfree.it
Cc: radar@myoffice.it, spiderac@myoffice.it, dodgio@myoffice.it
Subject: auguriDate: Wed, 25 Dic 2002 15:38:02 +0100
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
Content-Type: text/plain;
            charset="iso-8859-1"
X-Priority: 3
```

php

```
X-Mailer: Webmail
-----------------------------------------------------------------
corpo del messaggio
-----------------------------------------------------------------
Auguri di buon natale e felice anno nuovo
```

Una riga vuota distingue l'intestazione dal corpo del messaggio. **MIME-version** dice all'agente utente che sta ricevendo un messaggio **MIME**. Se tale intestazione non è presente, il messaggio sarà elaborato come un testo ASCII. **Content-Transfer-Encoding** dice all'agente utente come è stato trattato il corpo per la trasmissione attraverso la rete. Le codifiche più comuni per il testo, sono l'ASCII a 7 ed a 8bit. Il testo ASCII può essere trasportato direttamente dal protocollo di posta elettronica a patto che ciascuna linea non superi i 1000 caratteri. Invece il modo corretto per codificare i file binari è di utilizzare la codifica base64 chiamata anche protezione ASCII. In questo schema i ritorni carrello e gli "a capo" **(\r\n)** vengono ignorati, così da poter essere inseriti a piacere per mantenere le righe abbastanza brevi o comunque sotto il limite dei 1000 caratteri.

L'intestazione **Content-type** specifica all'agente utente la natura del corpo del messaggio. L'RFC 1521 definisce sette tipi di **Content-type**, ognuno dei quali ha uno o più sottotipi. Il tipo ed il sottotipo sono divisi dal carattere /.

I **Content-type** più comuni sono:

- **Text/plain** • **Text/html** • **Image/gif**
- **Image/jpeg** • **Audio/basic** • **Audio/wav**
- **Video/mpeg** • **Application/Octet-stream** • **Multipart/Mixed**

Gli ultimi due, **Application** e **Multipart**, sono i più interessanti.

Application viene definito per tutti quei formati che richiedono una elaborazione non prevista dagli altri tipi (es: **application /msword** per i file Word **(.doc)**, **application/zip** per i file **.zip,** e così via). Il sottotipo **octet-stream** indica una sequenza di byte non interpretabile e di conseguenza non associabile ad una specifica applicazione. In definitiva **Application/Octed-stream** è una classica intestazione utilizzata dal cliente di posta elettronica quando si cercano di spedire file per i quali sul sistema non esiste alcuna applicazione associata. Il tipo **Multipart** permette ad un messaggio di contenere più parti opportunamente divise da un codice univoco chiamato **boundary.** Il sottotipo **mixed** indica che le parti sono differenti una dall'altra. **Multipart/Mixed** è il **Content-type** che deve comparire nell'intestazione di ogni messaggio che contiene dei file allegati (o **attach**).

Vediamone un esempio:

```
from: joker@myoffice.it
to: emma@myoffice.it, adriano@interfree.it
Cc: radar@myoffice.it, spiderac@myoffice.it, dodgio@myoffice.it
Subject: auguri
Date: Wed, 25 Dic 2002 15:38:02 +0100
MIME-Version: 1.0
X-Priority: 3
X-Mailer: Webmail
Content-Type: multipart/mixed;
        boundary="5b1645599fa69d63b64d3178ca70f3b9"
This is a multipart message in MIME format.
--5b1645599fa69d63b64d3178ca70f3b9
Content-Type: text/plain;
                    charset="iso-8859-1"
Content-Transfer-Encoding: 8bit

 Auguri di buon natale e di felice anno nuovoJ
--5b1645599fa69d63b64d3178ca70f3b9
Content-Type: audio/wav
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
        filename="MaryChristmas.wav"
UkIGRtjZAABXQVZFZm10IBAAAAABAAIAIIYAAIhYAQAEABAAZGFOYYT
                                        ZAAACAAMACQAGAAcAAgA
FAAAAAwADAAEAAgD//wMABAABAAUAAQAFAA
                    IABQAAAAQABAADAAAACAACAAIABAAAAAIAAg
AAAAEAAAACAAIAAwACAAYAAQAGAAUABwACAAcAAQAJAAEABwA
                                        CAAIAAwACAAIABgABAAUA
BQAAAAIABgAAAAYAAQAGAA......
--5b1645599fa69d63b64d3178ca70f3b9--
--5b1645599fa69d63b64d3178ca70f3b9—
```

Questo esempio illustra il formato di una mime email contenente oltre al messaggio testuale (**Auguri di buon natale e di felice anno nuovo**) anche un allegato (**MaryChristmas-.wav**).

Come si può notare il **Content-type** è un **Multipart/Mixed.** Le sezioni del messaggio sono delimitate dal **boundary** definito nell'intestazione e preceduto dai caratteri **"--"**. Ogni sezione avrà una intestazione ed un corpo. Nell'intestazione saranno indicati il **Content-type**, il **Content-Transfer-Encoding** ed il **Content-Disposition**, mentre nel corpo sarà presente l'informazione opportunamente codificata (nel caso del file **wav** in base64). L'ultima sezione del messaggio si chiude con due linee vuote (**CRLF, CRLF**) ed il boundary preceduto e seguito dai caratteri **"--"**.

COS'È MD5

MD5 è un algoritmo per calcolare la funzione di hash di un file, ossia permette di generare un identificativo univoco di un file o di un flusso di dati. Gli algoritmi di hash sono utilizzati per verificare l'integrità dei dati trasmessi e quindi rivestono un ruolo fondamentale per la firma digitale.

COSTRUIAMO LA CLASSE SMTP

La classe **smtp.php** conterrà le proprietà:

- **$from:** un array che conterrà il nominativo e l'email del mittente.
- **$to, $cc, $ccn:** stringhe utilizzate per memorizzare rispettivamente l'indirizzo, o gli indirizzi (separati da virgole), dei destinatari, dei destinatari in copia carbone ed in copia carbone nascosta.

- **$boundary:** sarà settato nel caso di email che contengono allegati.

A questo punto si rende necessario realizzare un metodo (**add_attachment**) per accodare all'array $**parts** tutte le informazioni sugli allegati. I parametri in ingresso a questa funzione saranno: il nome del file, il **content-type**, il **content-transfer-encoding** ed il file (opportunamente caricato in una variabile). Questo sistema di funzionamento torna utile nel caso in cui l'allegato non è un fisicamente conservato su disco ma è un file creato dinamicamente dal PHP (come ad esempio nel caso dei **pdf**). Però è possibile, omettendo il parametro relativo al contenuto (**$message**) ed indicando un URL per il nome (**$file**), demandare alla funzione il compito di recuperare il file.
Altro metodo dedicato alla costruzione della email è **build_headers()**, che provvederà, in base alle proprietà, a definire l'header del messaggio: ogni stringa indicante i destinatari (separati da virgole), i destinatari in copia carbone ed in copia carbone nascosta, sarà esplosa in un array e gli indirizzi email saranno aggiunti all'intestazione. Se l'**array parts** contiene elementi (ossia ci sono allegati), sarà calcolato un id univoco ed assegnato al boundary presente nell'intestazione **Content-type: Multipart/Mixed.** Ci siamo!!! L'email è pronta per essere spedita.
A questo punto è necessario realizzare i metodi per gestire la comunicazione con il server SMTP.

RFC

Gli RFC (Request For Comments) sono documenti ufficiali che definiscono tutti i protocolli, le procedure ed i funzionamenti di ogni "parte" di Internet. Essi costituiscono l'ossatura fondamentale del funzionamento della rete e naturalmente, come il titolo stesso dichiara, sono di pubblico dominio. Ad oggi esistono oltre 3000 RFC.

SMTP- SIMPLE MAIL TRANSFER-PROTOCOL

Per poter comunicare con il server SMTP è necessario stabilire una connessione TCP (tramite i socket) alla porta 25 del computer sul quale il server è in esecuzione. Dopo che il cliente ha stabilito una connessione, il server si identifica spedendo una linea di testo che inizia con un codice numerico. In base al codice, il cliente riesce a capire se il server è pronto o meno a ricevere la posta. Se la risposta è positiva (codice 220) il cliente deve identificarsi al demone SMTP inoltrando il comando **HELO <domain name>**. Vediamo come avviene una semplice comunicazione tra un cliente ed un server smtp per la spedizione di una mime mail:

```
220 SMTP Server Ready
helo myoffice.it
250 g'day eh!
  mail from: joker@myoffice.it
250 'aight sender's cool
  rcpt to: amici@libero.it
250 ok
  rcpt to: pippo@pippo.it
250 ok
```

```
  data
354 go
  From: "antonio" <joker@myoffice.it>
  To: amici@libero.it
  Cc: pippo@pippo.it
  Subject: Fortune J
  Mime-Version: 1.0
  X-Mailer: Webmail
  X-priority: 1
  Content-Type: text/plain;
          charset="iso-8859-1"
  Content-Transfer-Encoding: 8bit
  "C'È vero progresso solo quando i vantaggi di una nuova
                         tecnologia diventano per tutti"
250 ok
```

Per stabilire una connessione con un server SMTP utilizziamo la funzione PHP **fsockopen**. Successivamente la funzione **check_ answer** legge la risposta del server e controlla se il codice numerico è uguale al parametro passato alla funzione. In caso contrario il processo sarà terminato e visualizzato il messaggio d'errore. Se la connessione è stata attivata correttamente è possibile spedire la nostra email tramite la funzione **sendmail()**, che si preoccuperà inoltre, se il boundary è stato settato dalla funzione **build_header()**, di invocare il metodo **send_multipart()** in modo da spedire anche gli allegati. La funzione **sendmail()** si conclude inviando al server una riga vuota seguita da un "." ed un CRLF. Quest'ultima operazione indicherà al demone SMTP che la fase di costruzione dell'email è finita e che questa può essere inoltrata verso le caselle di posta elettronica dei destinatari. **send_multipart()** è senza dubbio la funzione più interessante della nostra classe, in quanto, facendo riferimento alle informazioni contenute nell'array parts, provvederà ad inviare gli allegati. Se il campo "Message" dell'array è vuoto, il file sarà recuperato dalla posizione (URL o path) indicata nel "Filename". I file binari saranno codificati in base64 e, dopo aver aggiunto ogni 76 caratteri un CRLF (\r\n), spediti al server SMTP. I file di testo saranno spediti in chiaro, e per ottimizzare l'occupazione di memoria principale (nel caso in cui un file di testo deve essere recuperato), saranno inoltrati al server SMTP man mano che vengono letti (ogni 4096 byte).
L'ultima sezione terminerà inviando al server SMTP due righe vuote seguite dal boundary di chiusura:

```
"--$this->boundary"--".
```

CONCLUSIONI

In questo articolo si è mostrato come in PHP è possibile (tramite i socket) diventare clienti di una applicazione server presente su Internet. Nel caso specifico abbiamo stabilito una connessione TCP ad un server SMTP al fine di negoziare la spedizione di email eventualmente corredate da allegati.

Antonio Perri

La sicurezza del software prima parte

di Corrado Giustozzi

Un vecchio e feroce adagio dell'informatica dice, più o meno, che se gli ingegneri costruissero i palazzi come i programmatori scrivono il software, il primo picchio di passaggio potrebbe distruggere una città. È un'esagerazione, certo, un paradosso... ma, come tutti i paradossi, racchiude in sé una grande verità. Ci piaccia o no, la maggior parte del software in giro per il mondo è davvero troppo inaffidabile, ossia vulnerabile nei confronti di errori accidentali o di attacchi sistematici; ed in alcuni casi ciò comporta reali rischi per l'organizzazione che utilizza quel software o per i suoi utenti.
La Società dell'Informazione, massima realizzazione di millenni di evoluzione della civiltà occidentale, sembra dunque più un colosso dai piedi d'argilla che un sistema infrastrutturale stabile e sicuro; e anche se la tendenza verso la virtualizzazione dei rapporti sociali sembra ormai inarrestabile, sono in molti a dubitare dell'opportunità di affidare del tutto a sistemi tecnologici come quelli attuali la gestione della complessa rete di relazioni che costituisce il flusso vitale sul quale poggiano le realtà sociali, economiche, produttive, organizzative ed amministrative delle società contemporanee.
Certo la complessità dei sistemi non gioca a favore della loro affidabilità: ma il problema vero non è legato solo alle dimensioni dei sistemi quanto al loro intrinseco processo realizzativo, e più precisamente alla possibilità (o impossibilità...) di realizzare software robusto e sicuro oltre che corretto.

EQUILIBRI INSTABILI

Diversi sono i fattori critici che concorrono ad alimentare la spirale della vulnerabilità dei sistemi: l'ingestibilità del software oltre una certa soglia di complessità, la crescente tendenza verso l'utilizzo di sistemi "off the shelf" a bassa affidabilità anche nella realizzazione di infrastrutture critiche, l'intrinseca immaturità di talune tecnologie digitali, la modalità di progettazione spesso troppo approssimativa o "ingenua" persino nei sistemi cruciali. Di solito tutto funziona bene perché le condizioni in cui i vari sottosistemi si trovano ad operare sono quelle nominali, e le tolleranze di esercizio sono ampie: ma esiste sempre la possibilità che qualcosa collassi clamorosamente non appena un evento imprevisto porti il sistema al di fuori dello stato di equilibrio.
Il rischio, in un sistema molto complesso, è che al suo interno si inneschi un micidiale processo di "retroazione positiva": ossia che la reazione automatica ad una perturbazione vada in realtà nel verso sbagliato e destabilizzi ulteriormente la situazione. In questo caso ogni nuova reazione del sistema allo squilibrio peggiorerà ulteriormente le cose, instaurando un circolo vizioso che porterà ben presto il sistema stesso al di fuori delle specifiche di funzionamento e finirà per renderlo inoperativo. Ma non è finita qui: a questo punto è infatti probabile che si inneschi anche una reazione a catena verso i sistemi collegati i quali, accorgendosi del malfunzionamento del primo sistema (o, peggio ancora, non accorgendosene e continuando a prendere per buoni i suoi output...), nel tentativo di reagire all'anomalia potrebbero a loro volta fallire.
Situazioni del genere si sono già verificate in passato, ed hanno rischiato di causare catastrofi estremamente significative. Il famoso "venerdì nero" di qualche anno fa alla Borsa di New York, ad esempio, fu provocato proprio dai meccanismi automatici di controllo delle transazioni che, in teoria, avevano invece lo scopo di salvaguardare l'andamento dei mercati: un inaspettato ma non allarmante ribasso di alcuni titoli-guida provocò l'attivazione di alcune procedure di "protezione" che procedettero a vendere i titoli in calo; ma l'effetto di queste vendite fu una perdita più generalizzata e consistente, alla quale le procedure automatiche reagirono con ulteriori vendite sempre più massicce. In pochi minuti la Borsa giunse quasi al crollo. Quella lezione ha insegnato che i sistemi automatici di controllo non devono essere troppo reattivi per non provocare pericolosi "effetti valanga", e da allora i meccanismi di stabilizzazione delle Borse sono stati rivisti e corretti; ma questo non assicura che essi sapranno reagire correttamente ad ogni situazione nuova ed anomala che dovesse presentarsi in futuro.

SOFTWARE CORRETTO E SOFTWARE ROBUSTO

Purtroppo il programmatore quadratico medio di so-

lito non affronta lo sviluppo di un'applicazione in un'ottica "paranoica", ma dà per scontato che il suo software funzionerà in un contesto grosso modo rispondente alle specifiche; il che è un errore, in quanto non basta che un programma sia corretto per ritenere che sia anche affidabile.

Il fatto è che il software è un oggetto relativamente giovane e, checché se ne dica, non è un prodotto industriale: pertanto non sono state ancora sviluppate (e a maggior ragione maturate) dalla comunità informatica quelle necessarie conoscenze, sia teoriche sia pratiche, che consentirebbero di realizzarlo in modo accettabilmente e dimostrabilmente affidabile. La programmazione è ancora oggi più un'arte che una scienza, e decenni di metodologie formali di sviluppo o di programmazione ad oggetti hanno avuto impatti solo sulle modalità di conduzione dei progetti software dalle dimensioni stratosferiche, non certo dei progetti medio-piccoli che generalmente sono quelli nei quali sono coinvolti i comuni mortali.

CORRADO GIUSTOZZI

Romano doc, classe 1959. Giornalista scientifico ed esperto di sicurezza informatica. Ha iniziato a scrivere di computer nel 1979, e da allora ha pubblicato oltre mille articoli e tre libri. È stato direttore tecnico e vicedirettore di MCmicrocomputer, ed ha fondato e diretto Byte Italia. Attivo sin dal 1985 sui temi della sicurezza e della crittografia, ha tra l'altro collaborato col Comando Generale dell'Arma dei Carabinieri in indagini sulla criminalità informatica. Attualmente è security evangelist presso Secure Edge di Roma. La cosa più seria che ha fatto è sviluppare per l'Istituto dell'Enciclopedia Italiana il software di giochi linguistici del Vocabolario Italiano Treccani su CD-ROM. Il suo Web è http://www.nightgaunt.org

In generale è già tanto quando un programmatore applica nel suo lavoro i principi della cosiddetta programmazione difensiva. Con questo termine generico si indica tutta quella serie di tecniche, spesso non formalizzate ed essenzialmente euristiche, ossia lasciate alla sensibilità ed all'esperienza del singolo programmatore, mediante le quali si cerca di far sì che il programma che si sta sviluppando sia non solo corretto, ovvero lavori secondo le specifiche, ma anche robusto, ovvero riesca a mantenere un funzionamento corretto anche in presenza di errori o di situazioni impreviste. Un programmatore con un sano atteggiamento difensivo, ad esempio, non si fiderà ciecamente dell'input passatogli dall'utente ma provvederà a validarlo da un punto di vista sintattico prima di elaborarlo; provvederà a verificare i risultati ritornati dalle chiamate di sistema per catturare in tempo eventuali errori di runtime; disseminerà il suo codice di assert o costrutti analoghi, e così via. Tutto sommato, come dice un altro assioma dell'informatica, la programmazione è quella disciplina dove l'impossibile accade regolarmente, dove le costanti non lo sono e le variabili non mutano, per cui è meglio diffidare di quasi tutto ciò che succede!

Purtroppo però neppure la programmazione difensiva serve a proteggersi contro tutti i possibili accidenti che potrebbero accadere ad un programma. Il cosiddetto Secondo Postulato di Troutman, una delle applicazioni all'informatica della Legge di Murphy, recita infatti: "se il programma è stato concepito in modo tale che i dati incorretti siano rifiutati, ci sarà sempre un idiota abbastanza ingegnoso il quale troverà il metodo per farli passare lo stesso". Molti dunque rinunciano in partenza, confidando del fatto che tanto il programma funzionerà bene nella maggior parte dei casi anche in assenza di specifici meccanismi di verifica, i quali tanto non potrebbero riuscire a intercettare e rendere inoffensiva ogni fonte di disturbo.

SOFTWARE ROBUSTO E SOFTWARE SICURO

Un programma corretto, dunque, non è necessariamente robusto; e a maggior ragione non è necessariamente sicuro. La differenza è sottile ma fondamentale: un software robusto resiste ad errori grossolani negli input ed a situazioni anomale generate per accidente, ma generalmente soccombe ad un attacco mirato che ha come scopo la sua disabilitazione o l'alterazione deliberata delle sue caratteristiche di funzionamento; un software sicuro resiste anche a questo, o se fallisce lo fa almeno in modo tale da non offrire il fianco a danni più gravi.

Non è facile scrivere software sicuro; tuttavia ci si può provare, almeno nei limiti di una sana paranoia e compatibilmente con le esigenze e gli ambiti operativi nei quali ci si trova ad operare. Il software di una centrale nucleare o di un firewall, ovviamente, devono essere molto sicuri oltre che robusti; ma anche quello di un gioco multiutente, se vogliamo ad esempio evitare che un intruso, sfruttando ad arte qualche bug o "feature" del gioco, possa accedere ai file memorizzati sui PC degli altri giocatori. E gli esempi potrebbero essere quasi infiniti.

Il prossimo mese vedremo dunque qualche esempio pratico di software corretto ma non robusto e non sicuro, discutendo come banali bug o semplici mancanze di controlli possano in determinate situazioni diventare brecce di sicurezza straordinariamente gravi. Dopo di che non saremo automaticamente in grado di scrivere software avionico per i caccia militari, ma forse guarderemo con occhi diversi il processo di sviluppo di applicazioni solo apparentemente ovvie e banali, come ad esempio un semplice script di controllo degli accessi ad una pagina Web protetta.